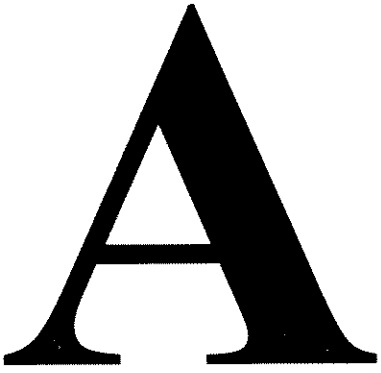


US006505328B1

(12) **United States Patent**
Van Ginneken et al.

(10) **Patent No.: US 6,505,328 B1**
(45) **Date of Patent: Jan. 7, 2003**

(54) **METHOD FOR STORING MULTIPLE LEVELS OF DESIGN DATA IN A COMMON DATABASE**

(75) Inventors: **Lukas P. P. P. Van Ginneken**, San Jose, CA (US); **Patrick R. Groeneveld**, San Jose, CA (US); **Wilhelmus J. M. Philipsen**, Phoenix, AZ (US)

(73) Assignee: **Magma Design Automation, Inc.**, Cupertino, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/300,540**

(22) Filed: **Apr. 27, 1999**

(51) **Int. Cl.**[7] **G06F 17/50**
(52) **U.S. Cl.** **716/7**; 716/8; 716/12
(58) **Field of Search** 716/1–21

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,187,668	A	*	2/1993	Okude et al.	716/8
5,212,650	A	*	5/1993	Hooper et al.	716/18
5,313,615	A	*	5/1994	Newman et al.	716/11
5,432,707	A	*	7/1995	Leung	716/2
5,452,226	A	*	9/1995	Hooper et al.	716/18
5,487,018	A	*	1/1996	Loos et al.	716/11
5,519,627	A	*	5/1996	Mahmood et al.	716/18
5,541,849	A	*	7/1996	Rostoker et al.	716/18
5,623,417	A	*	4/1997	Iwasaki et al.	716/18
5,666,288	A	*	9/1997	Jones et al.	716/17
5,696,693	A	*	12/1997	Aubel et al.	716/8
5,699,265	A	*	12/1997	Scepanovic et al.	716/10
5,726,902	A	*	3/1998	Mahmood et al.	716/6
5,727,187	A	*	3/1998	Lemche et al.	716/18
5,757,657	A		5/1998	Hathaway et al.	710/9
5,761,664	A		6/1998	Sayah et al.	707/100
5,764,534	A		6/1998	Goetting	716/11
5,818,729	A	*	10/1998	Wang et al.	716/9
5,841,663	A	*	11/1998	Sharma et al.	716/18
5,864,487	A	*	1/1999	Merryman et al.	716/6
5,956,497	A	*	9/1999	Ratzel et al.	716/1

(List continued on next page.)

OTHER PUBLICATIONS

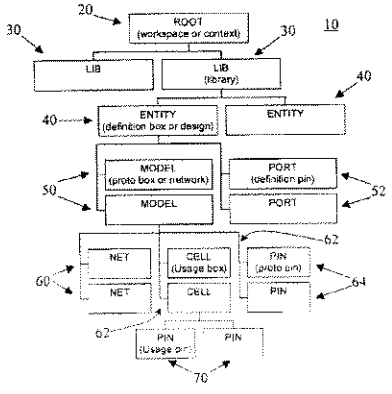
Dutt ("Generic component library characterization for high level synthesis", Proceedings of the Fourth CSI/IEEE International Symposium on VLSI Design, 1991, Jan. 4, 1991, pp. 5–10).*
Dion, J. and Monier, L.M.; "Countour: A Title–based Gridless Router," *WRL Research Report 3/95*, Digital Western Research Laboratory.
Dion, J. and Monier, L.M.; "Recursive Layout Generation," *WRL Research Report 2/95*, Digital Western Research Laboratory.
Hwang., J., et al., "Generating layouts for self–emplementing modules"; Intl Workshop on Field Programmable Logic and Applications, FPGAS, GB, Abingdon, Aug. 31, 1998, pp. 525–529.
Singhal, A., et al., "Object oriented data modeling for VLSI/CAD," Proc. of the 8th Intl Conf. on VLSI Design, New Delhi, India, Jan. 4–7, 1995, pp. 25–29.
Fcanha, H.S.;, "Data astructtures for physical representation of VLSI," Software Eng'g Journal, GB, IEE. London, vol. 5, No. 6, Nov. 1, 1990.

Primary Examiner—Matthew Smith
Assistant Examiner—Phallaka Kik
(74) *Attorney, Agent, or Firm*—Pillsbury Winthrop LLP

(57) **ABSTRACT**

An automated logic circuit design system uses a common database to store design data at different states of the design process, including data-flow graphs, netlists and layout descriptions. In this way, the need to translate circuit descriptions between tools is eliminated, thus leading to increased speed, flexibility and integration. The common database includes entities, models, cells, pins, busses and nets. The data-flow graphs are stored as graphs, the nodes in a graph as cells, and the edges as busses. Physical design data is available by storing the cells in a model in a KD tree. This allows queries on cells in the netlist located in the layout within arbitrary areas.

17 Claims, 4 Drawing Sheets



U.S. PATENT DOCUMENTS

5,960,184	A *	9/1999	Cleereman et al.	716/3
6,026,228	A *	2/2000	Imai et al.	716/188
6,080,201	A *	6/2000	Hojat et al.	703/14
6,145,117	A *	11/2000	Eng	716/18
6,154,874	A *	11/2000	Scepanovic et al.	716/13
6,216,258	B1 *	4/2001	Mohan et al.	716/17
6,263,483	B1 *	7/2001	Dupenloup	716/18
6,289,489	B1 *	9/2001	Bold et al.	716/1
6,308,309	B1 *	10/2001	Gan et al.	716/8

* cited by examiner

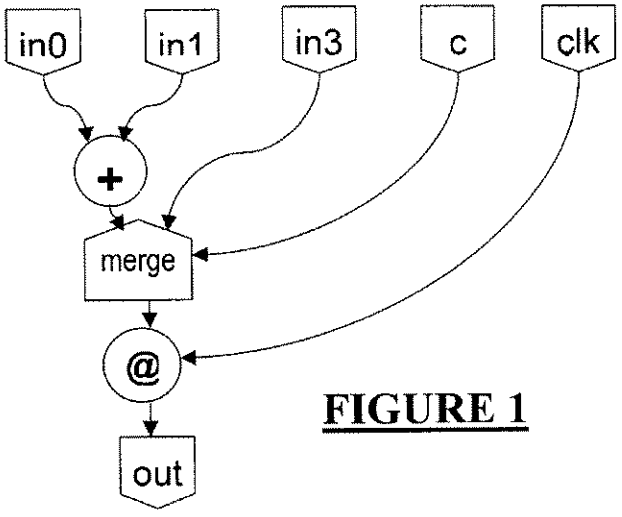


FIGURE 1

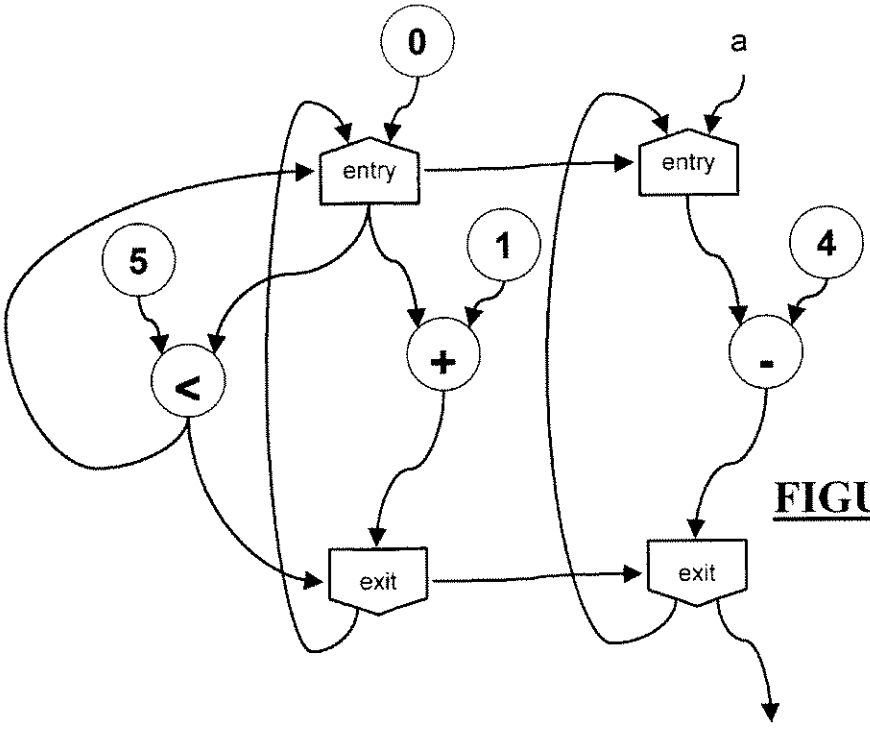


FIGURE 3

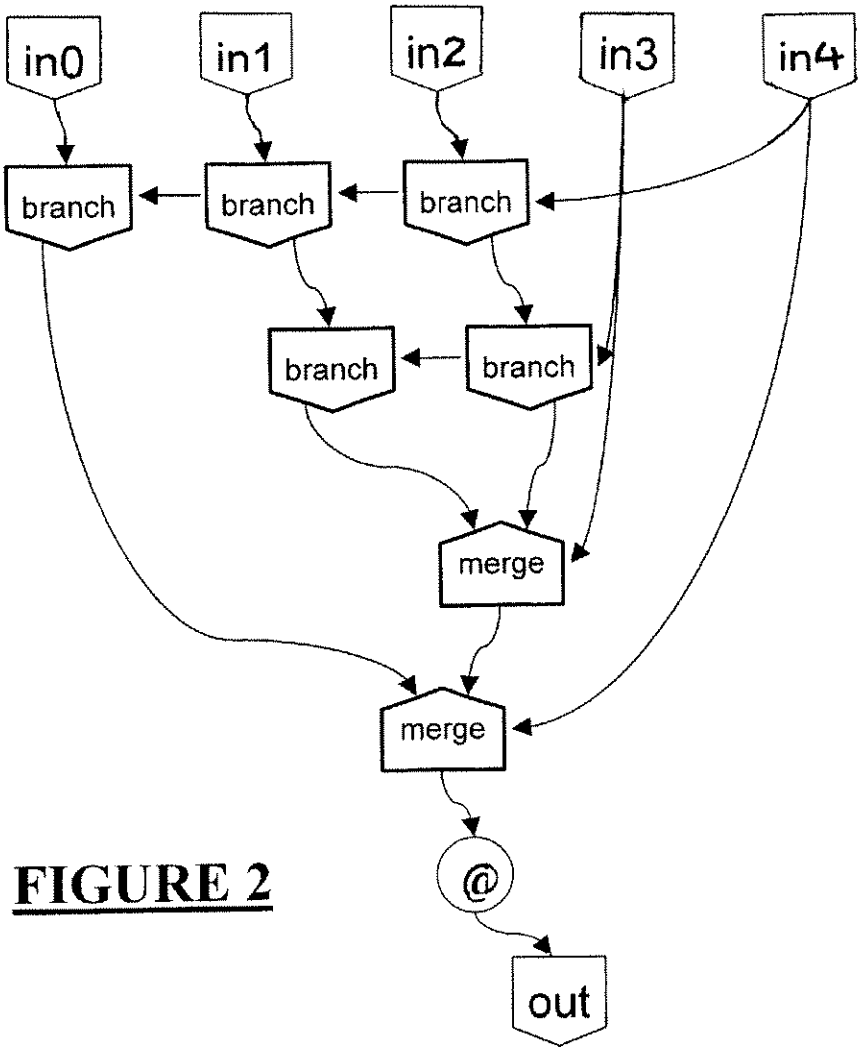

in0
in1
in2
in3
in4
branch
branch
branch
branch
branch
merge
merge
@
out

FIGURE 2

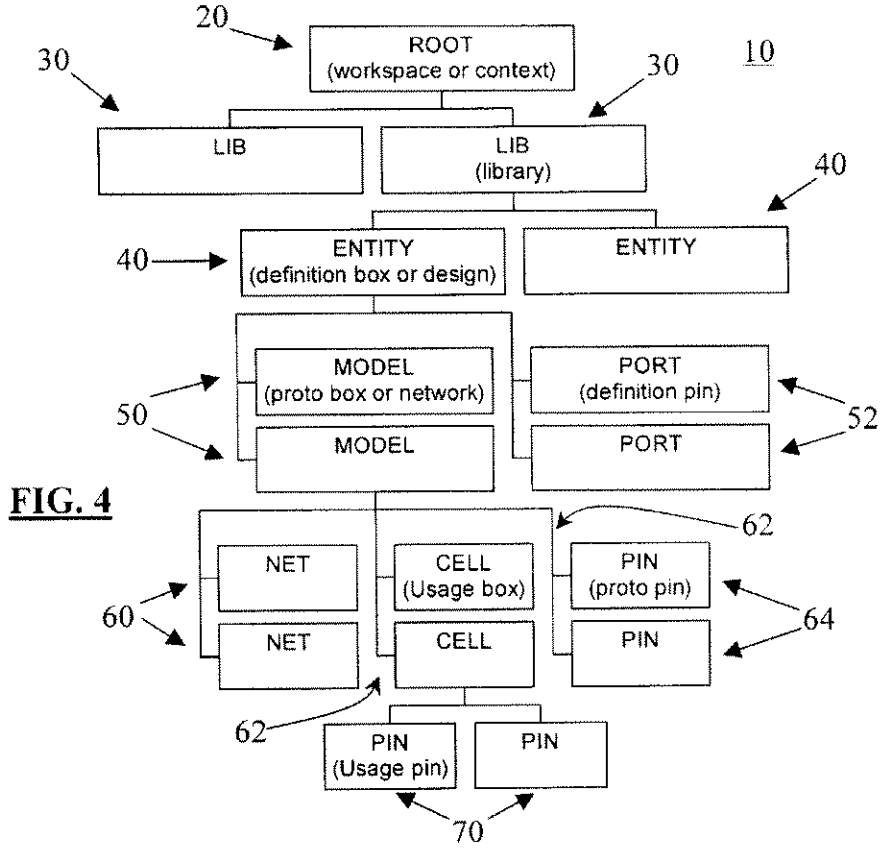


FIG. 4

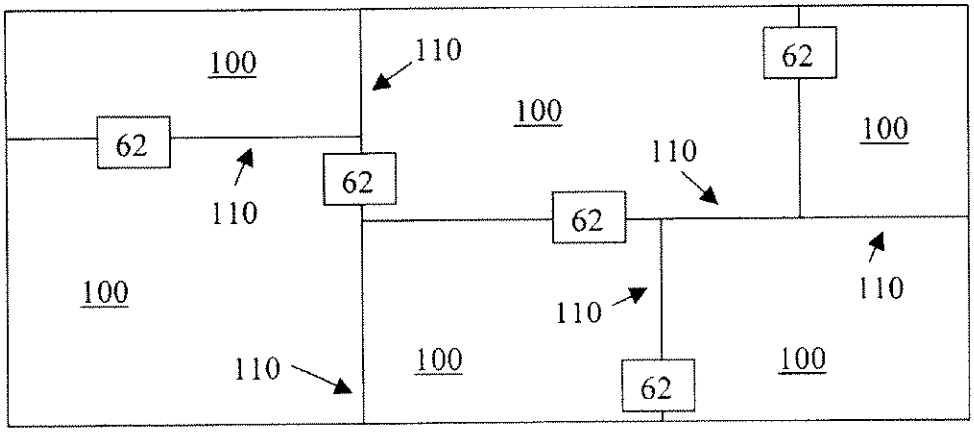


FIGURE 5

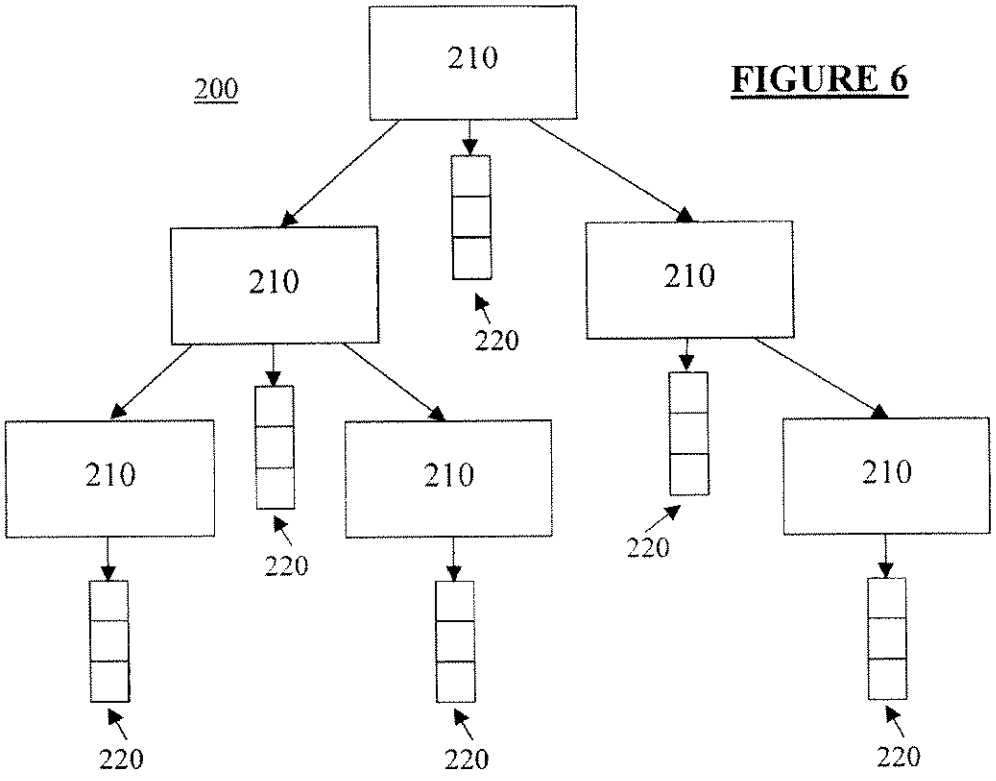


FIGURE 6

METHOD FOR STORING MULTIPLE LEVELS OF DESIGN DATA IN A COMMON DATABASE

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention is directed to digital logic design systems. More particularly, the invention is directed to automated digital logic synthesis and placement systems.

2. Background of the Related Art

Prior art computer aided design (CAD) systems for the design of integrated circuits and the like assist in the design thereof by providing a user with a set of software tools running on a computer. In the prior art, the process of designing an integrated circuit on a typical CAD system was done in several discrete steps using different software tools.

First, a schematic diagram of the integrated circuit is entered interactively to produce a digital representation of the integrated circuit elements and their interconnections. This representation may initially be in a hardware description language such as Verilog and then translated into a register transfer level (RTL) description in terms of predesigned functional blocks, such as memories and registers. This may take the form of a data structure called a net list.

Next, a logic compiler receives the net list and, using a component database, puts all of the information necessary for layout, verification and simulation into object files whose formats are optimized specifically for those functions.

Afterwards, a logic verifier checks the schematic for design errors, such as multiple outputs connected together, overloaded signal paths, etc., and generates error indications if any such design problems exist. In many cases, the IC designer improperly connected or improperly placed a physical item within one or more cells. In this case, these errors are flagged to the IC designer so that the layout cells may be fixed so that the layout cells perform their proper logical operation. Also, the verification process checks the hand-laid-out cells to determine if a plurality of design rules have been observed. Design rules are provided to integrated circuit designers to ensure that a part can be manufactured with greater yield. Most design rules include hundreds of parameters and, for example, include pitch between metal lines, spacing between diffusion regions in the substrate, sizes of conductive regions to ensure proper contacting without electrical short circuiting, minimum widths of conductive regions, pad sizes, and the like. If a design rule violation is identified, this violation is flagged to the IC designer so that the IC designer can properly correct the cells so that the cells are in accordance with the design rules.

Then, using a simulator the user of the CAD system prepares a list of vectors representing real input values to be applied to the simulation model of the integrated circuit. This representation is translated into a form which is best suited to simulation. This representation of the integrated circuit is then operated upon by the simulator which produces numerical outputs analogous to the response of a real circuit with the same inputs applied. By viewing the simulation results, the user may then determine if the represented circuit will perform correctly when it is constructed. If not, he or she may re-edit the schematic of the integrated circuit, re-compile and re-simulate. This process is performed iteratively until the user is satisfied that the design of the integrated circuit is correct.

Then, the human IC designer presents as input to a logic synthesis tool a cell library and a behavioral model. The behavioral circuit model is typically a file in memory which looks very similar to a computer program. The behavioral circuit model contains instructions which define logically the operation of the integrated circuit. The logic synthesis tool receives as input the instructions from the RTL circuit model (i.e., Verilog or VHDL) and the library cells from the library. The synthesis tool maps the instructions from the behavioral circuit model to one or more logic cells from the library to transform the behavioral circuit model to a gate schematic net list of interconnected cells. A gate schematic net list is a data base having interconnected logic cells which perform a logical function in accordance with the behavioral circuit model instructions. Once the gate schematic net list is formed, it is provided to a place and route tool.

The place and route tool is used to access the gate schematic net list and the library cells to position the cells of the gate schematic net list in a two-dimensional format within a surface area of an integrated circuit die perimeter. The output of the place and route step is a two-dimensional physical design file which indicates the layout interconnection and two-dimensional IC physical arrangements of all gates/cells within the gate schematic net list.

According to the above prior art method, a separate internal data structure is used for each tool. This is because the tools are rarely if ever written by the same group; thus, the internal database representation for each tools is likely to differ from that of the other tools. Also, the most appropriate database implementation for the integrated circuit depends on the phase of the design process in which it is being used. For example, linked lists are commonly used to store cells in a netlist because that is the most obvious solution for logic synthesis purposes. In contrast, a KD tree is a more appropriate database format for the place and route tool.

This is time-consuming and processor-intensive (circuit specifications must be translated from one database format to another and another during the development process), disk-intensive (multiple databases each specifying the same circuit in different forms must be stored) and fragmented (tools cannot use the outputs of other tools, and a change to the circuit made by one tool is not reflected in the databases of the other tools).

SUMMARY OF THE INVENTION

The present invention has been made with the above problems of the prior art in mind, and a first object of the present invention is to provide a system for automated logic circuit design which is capable of storing and utilizing multiple levels of design data in a common database.

Another object of the present invention is to provide a system for automated logic circuit design which eliminates the need for translation of circuit descriptions between different design tools.

A further object of the present invention is to provide a system for automated logic circuit design which allows the output of tools in the design suite to be used by other tools.

Yet another object of the present invention is to provide a system for automated logic circuit design which allows design tools or the user to make area queries, i.e., a selection of a subset of objects based on their physical position, at various stages in the design process.

A still further object of the present invention is to provide a system for automated logic circuit design which permits the use of global simulation tools such as timing engines across all levels of design abstraction.

Another object of the present invention is to provide a system for automated logic circuit design which presents a unified model for timing, synthesis, placement and routing.

A further object of the present invention is to provide a system for automated logic circuit design which has high storage and run-time efficiency.

A still further object of the present invention is to provide a system for automated logic circuit design which has a consistent and easy to use programming interface.

A still further object of the present invention is to provide a system for automated logic circuit design which has an interface which is not dependent on other include files.

A further object of the present invention is to provide a system for automated logic circuit design which uses an object-oriented C++ programming style.

The above objects are achieved according to an aspect of the invention by providing an automated logic circuit design system which uses a common database to store design data at different states of the design process, including data-flow graphs, netlists and layout descriptions. In this way, the need to translate circuit descriptions between tools is eliminated, thus leading to increased speed, flexibility and integration. The common database includes entities, models, cells, pins, busses and nets. The data-flow graphs are stored as graphs, the nodes in a graph as cells, and the edges as busses. Physical design data is available by storing the cells in a model in a KD tree. This allows queries on cells in the netlist located in the layout within arbitrary areas.

BRIEF DESCRIPTION OF THE DRAWINGS

These and other objects, features, and advantages of the present invention are better understood by reading the following detailed description of the preferred embodiment, taken in conjunction with the accompanying drawings, in which:

FIGS. **1**–**3** are dataflow diagrams of a circuit structure according to a preferred embodiment of the present invention;

FIG. **4** is a block diagram of the structure of a data model according to the present invention; and

FIGS. **5** and **6** are diagrams of the partitioning of a chip in correspondence with the data model.

DETAILED DESCRIPTION OF THE PRESENTLY PREFERRED EXEMPLARY EMBODIMENT

First, given a Verilog description of a circuit to be developed, the Verilog must be parsed to generate a data flow graph suitable for implementation in the data model. RTL parsers known in the art are preferably used for this purpose. The output from the RTL parser is a Verilog parse tree which is used to generate the data flow graph. Although well-known in the art, the structure of the parse tree is relatively complicated and, since detailed knowledge of it is not necessary for an understanding of the present invention, further description of the parse tree will be omitted for simplicity and brevity.

FIG. **1** shows an example of translation of the Verilog source code

```
always @(posedge clk)
begin
  out=in1+in2;
  if (c)
    out=in3;
end
```

into data flow elements. Here, in**0**, in**1**, in**2**, c and clk are input ports of an Entity (described below) and out is an output port of the Entity. An adder (an example of a Cell as described bellow) adds the values at Ports in**0** and in**1** and supplies the result to a merge block (another example of a Cell). If the value at Port c represents a logical true, the merge block supplies the value at Port in**2** to a delay block (again, a Cell); if the value at Port c represents a logical false, the merge block supplies the output of the adder to the delay block. On the positive-going edge of the signal at Port clk, the delay block provides the value on its input to the output port out. The data flow graph having been generated, it may then be stored in the data model.

Before describing the data model in more detail, a few more examples are in order. FIG. **2** shows an example of the data graph resulting from the Verilog code

```
if (c1) begin
  out=in0;
end else begin
  if (c2) begin
    out=in1
  end else begin
    out=in2
  end
end
```

Here, if the value at input Port c**1** is a logical true, the branch module connected to input Port in**0** is enabled and provides its output to a merge module which is also enabled when c**1** is true. If c**1** is not true, the branch modules connected to Ports in**1** and in**2** are enabled to provide their outputs to other branch modules. One of the modules in this second tier is enabled when the value at input Port c**2** is true and provides its output to another merge block. The other of the modules in the second tier is enabled when c**2** is false and provides its output to the other merge block. Depending on the value of c**2**, one of those outputs is provided to the first merge block, and depending on the value of c**1**, one of the output is provided to the output port out via the delay element.

Finally, repetitive structures such as for, while and forever loops can be implemented in the data flow graph. FIG. **3** shows an example of the data flow graph implementation of the Verilog code

```
integer i;
for (i=0; i<5; i++)
begin
  a=a-4
end
```

Here, an entry node initializes a loop index with the value 0, adds 1 to the index and checks to see if the index is less than 5. If so, an exit node loops back for another pass. In tandem with this loop, an input variable a is received through another entry node and 4 is subtracted from it on each pass through the loop. When looping ceases because the index has reached 5, the current value of the input variable is presented at the exit node.

Certain optimizations can be performed on the data flow graph. For example, in the above loop structure the loop can be unrolled. That is, the graph portion representing the body of the loop can be replicated five times and the graph portion representing the loop index can be eliminated. For timing estimations and the like, virtual loop unrolling can be performed by estimating the number of iterations through the loop and using that number as a multiplier in delay calculations; the actual circuit replications can be done later.

Once the Verilog source is converted to a data flow graph, it can be implemented in the data model. Preferably, the data model is implemented using the C++ programming language

or a similar object-oriented language. Since the construction, accessing and destruction of objects in such languages is well-known in the art, examples of specific commands for performing these operations will be omitted for brevity.

The topmost object in the data model **10** (shown in FIG. **4**) is the Root object **20**. The Root object **20** owns all other objects **30–70** and serves as a base to which everything else is attached. Also, the root **20** accommodates global attributes which are shared by all objects **20–70**.

At the next level of the data model **10** is the Library object **30**. Library objects **30** are used to organize entities **40**. The preferred embodiment of the present invention includes at least two Libraries **30**. The first stores information on the technology library to which the circuit under development will be mapped, and the second stores information on the circuit itself.

Next is the Entity object **40**. An Entity **40** defines the interface of a design; that is, the set of ports **52** that the Entity **40** has. An Entity **40** may own Port objects **52**. A Port **52** is used to represent the pin-out of an entity **40**. Ports **52** cannot be individually created and destroyed, and can only be created when an Entity **40** is created. Each Port **52** has a direction (in, out, in/out) which is determined upon creation of the Entity **40**. This rigidity promotes consistency between the Entity **40**, its Models **50** and the Cells **62** bound to those Models **50**.

As noted above, Entities **40** own Models **50**. A Model **50** defines an implementation of an Entity **40**; thus, multiple Models **50** within an Entity **40** represent different implementations of that Entity **40**. Generally, these Models **50** are functionally identical to one another. For example, an Entity **40** in a technology Library **30** may have several Models **50** defining various eight bit adder cells having different power levels. Similarly, an Entity **40** in a target Library **30** may have several Models **50** which respectively define an abstract logic representation of a circuit, a gate-level implementation of it, a uniquefied representation, etc. The contents of each Model **50** is a net list of Nets **60**, Cells **62** and Model Pins **64**. All Models **50** and the Entity **40** have the same number of Pins **64** and Ports **52**, and the Ports **52** have the same direction in the Entity **40** and over all Models **50**; thus, it is relatively easy to replace one Model **50** with another from the same Entity **40**.

Below the Models **50** are Cell objects **62**. A Cell **62** represents a section of logic. Cells **62** may be primitive cells or non-primitive cells. Primitive Cells **62** have a predefined functionality associated with them. Preferably, the primitive Cells **62** include the following types:

CELL_AND—unlimited fan-in AND gate;

CELL_XOR—unlimited fan-in OR gate;

CELL_TRI—tri-state buffer

CELL_REG—sequential element

CELL_DC—don't care set

CELL_BREAK—break point cell; used to implement a "don't touch"; and

CELL_ONE—a constant one; an inverted or bubbled version is used for a constant zero.

In contrast to primitive Cells, the functionality of non-primitive Cells is defined by technology Models **50** to which they are bound. That is, a Cell **62** may describe a portion of the circuit under development and belong to a Model **50** in a target Library **30**. However, it will be associated with (preferably by pointing through a cell type attribute or the like) a Model **50** in a technology library **30** which defines its functionality and general characteristics.

Non-primitive Cells **62** may be created as bound Cells; alternatively, they may be created as unbound Cells and later bound to a Model **50**. This may be done by specifying the Cell's name; by specifying pin-to-pin correspondence vectors; and by binding the Cell **62** to an undefined Model **50** and later matching the Model **50** to an actual one. Additionally, a bound Cell **62** can be rebound to a different Model **50** within the same Entity **40**.

Each Cell **62** includes a number of parameters called members which specify certain features of the Cell **62**. These include the cell's name, a pointer to the technology Model **50** to which it is bound, a list of Pins **64** which it owns, its parent Entity **40**, and coordinates of the Cell **62** within the chip layout.

Net objects **60** make connections between pins. The pins may be Model pins **64** or Cell pins **70**. A Net **60** does not own Pins **64** and **70**, and deleting the Net **60** will leave the pins **64** and **70** disconnected. Pins **64** and **70** may be grouped into Busses **80** (in fact, every variable in the Verilog code will be represented as a Bus). Since Pins **64** and **70** are the most common object in almost any circuit representation, it is important to reduce the amount of storage for each Pin **64** and **70** as much as possible while maintaining easy accessibility. For this reason, Pins **64** and **70** are preferably stored in small arrays and associated with indices.

Nets **60** also have members, such as the Net's name, a list of Pins **64** and **70** which it connects, and a list of rectangles through which it passes in the placement layout. Pin members include the Pin's name, the Model **50** or Cell **62** to which it belongs, and the Net **60** to which it is connected.

Each object **20–70** may have a number of attributes. Each attribute has a name and a value of a type int, short, float, double, char* and void*. One example of an object attribute is an inversion attribute or "bubble" which specifies whether a Cell input or output (or Net **60**) is asserted high or low. Other examples of object attributes are object name, firing information, references to the Verilog code defining the object, etc.

Iterators are procedures used to access objects within the data model. As is known in the art, an iterator traverses the model and each time it is called, returns a pointer to the next object of a particular type. For example, a Model iterator would, when successively called, return pointers to each Model **50** within the data model. The preferred embodiment of the present invention provides "safe" and "unsafe" iterators, where unsafe iterators return all objects of the specified type, even if they have been added during the iteration process, and safe iterators omit objects added during the iteration. In this way, although the safe iterators are slightly slower than their unsafe counterparts, they can avoid program crashes, errors and exceptions, and other undesirable outcomes.

Before synthesis and timing can take place it is often necessary to uniquefy the data model. This involves binding each Cell **62** to its own individual technology Model **50**. This simplifies the synthesis process in that changes made to one technology Model **50** will affect only the Cell **62** which is bound to it, and no others. Also, after uniquefication it is possible to traverse the data model both up and down, since each object has a unique parent and child. Typically, uniquefication is done by making a copy of a technology Model **50** for each Cell **62** which is bound to it and associating one of the cells **62** to each copy.

After the data model has been uniquefied, it may be ungrouped, i.e., macro-level cells can be replaced with their primitive components. Alternatively, processes may handle the data model with virtual ungrouping by "looking through" the macro-level cells to process their primitive cell constituents.

With this understanding of the structure of the data model in mind, implementation of a Verilog-derived data flow graph in the data model will now be described. For each module in the Verilog description there will be one Entity **40** and one Model **50** (hereinafter collectively referred to as a graph). The ports for the Entity **40** correspond to the ports in the Verilog module. Ports **52** in the graph have a bit width, and there will be a separate Pin **64** and Net **60** (the group of Nets **60** for the Port **52** forming a Bus) in the graph for each Verilog port.

For each node in the Verilog module, a Cell **62** will be made in the graph. Initially the Cells **62** will be unbound. As described above, given the Cell type and the Pins **70** of the Cell **62**, a Model **50** for the Cell **62** to be bound can be generated later.

Each Model **50** is preferably implemented as a KD tree as follows. First, the circuit under development is divided into a number of sections each corresponding to a rectangular section **100** of the available chip area as shown in FIG. **5**. The partitioning of the circuit can be directed by the user; however, it is preferably automatically done by the system so that the circuit is evenly distributed over the entire chip area. Each node or leaf **210** of the KD tree **200** shown in FIG. **6** corresponds to a cutline **110** of the rectangles **100** and may have appended thereto a linked list **220** of all cells **62** which lie on that cutline **110**. Non-leaf nodes **210** in the KD tree **200** each have two child nodes **210**, with the left child **210** corresponding to the region of the chip on one side of the cutline **110** and the right child **210** corresponding to the region of the chip on the other side of the cutline **110**. Similarly, the child nodes **210** may have linked lists **220** of cells on their cutlines **110** and child nodes **210** of their own.

It should be noted that the leaf nodes **210** will contain most of the circuit information, since the non-leaf nodes **210** will only have information on those cells touching their corresponding cutline.

As noted above, the initial distribution of Cells **62** over the chip area is preferably done automatically by the system and in that case may be done through the use of various algorithms which will readily suggest themselves to those skilled in the art. The result of this process is a model with mostly logical information on its constituent elements but with a coarse framework of physical placement and routing information, e.g., cell areas, initial placements, etc. In later steps of the development process described below, the physical information will be refined and augmented within the original data model. In this way, the addition of rough physical layout information to the initial logical description enables the smooth transition of the circuit through the development process, thereby enabling sharing of tool outputs, use of common diagnostics and the like.

Further, once RTL synthesis is complete and the data model is flattened, it may be copied and used as a baseline for formal verification and the like. Since a common model structure is used, there is no need to translate the pre-logic synthesis version of the circuit into a format suitable for use by the verification tool.

As the development process progresses, the KD tree **200** may become unbalanced due to an excessive number of additions or deletions in one area, or due to poor initial distribution. This can be compensated for by manual rebalancing by the user or by a user-initiated procedure, but preferably is done automatically by the system.

Once the data model has been constructed in this way, it may be used for both logic synthesis, i.e., gate-level implementation, etc., and physical synthesis, i.e., placing and routing. This is because the data model includes all of the information necessary for logical synthesis operations, i.e., cell functionality, net connections, etc., as well as all information necessary for physical synthesis operations, i.e., areas, physical positions, etc.

Another advantage of the data model arises from its correspondence with the actual physical chip layout. Since each node of the KD tree **200** corresponds to a cutline **110** and has associated with it the cells on the cutline and information on where its child nodes are within the chip area, portions of the circuit in specific physical areas can be queried, tested and manipulated without the need to read the entire data model into active memory from disk storage, as is the case with prior art net lists. For example, assuming a user wanted to work with only the lower right hand corner of the chip, the system could traverse the KD tree to reach the topmost node corresponding to that area. Then, that node, its children, netlists and the like would be read into active memory from disk and manipulated. The user may even be able to manually direct placement of cutlines **110** at certain points to frame a particular area of interest. The system may then adjust the KD tree accordingly to accommodate the new arrangement. This area query technique is possible whether the circuit is in its final placement and routing stages or fresh from Verilog synthesis.

Although only a portion of the entire data model need be read into memory, the complete set of Nets **60** is typically maintained in memory. This is because the Nets **60** are necessary for purposes such as delay estimation and the like that are performed frequently, and it is easier to retain all Nets **60** in memory rather than repeatedly read them into memory. Thus, once a specific area has been designated for querying, the Nets **60** corresponding to that area must be identified. This is done by identifying the Nets **60** connected to each of the Pins **64**, **70** within the selected area. The remaining Nets **60** can be eliminated from consideration during the area query. Nets **60** which have some, but not all, Pins **64**, **70** within the query area can have the missing pins represented by a stub pin. Finally, Nets **60** which have all of their pins within the query area can be handled as are other objects within the selected area.

Further, during the area query process, Nets **60** which are entirely contained within the selected area can be optimized out or otherwise modified; however those nets having portions outside the query area, i.e., those with stub pins, cannot, since the effect of modification of elimination of these Nets **60** on the remaining circuit portions is unpredictable.

Further, since the logical and physical aspects of the circuit are integrated into a single data model from the start, deviations from the classic logical synthesis/physical synthesis partition can be made. For example, the inclusion of buffers for load handling and timing purposes is normally done as part of the logical synthesis process; however, using a common data model for all aspects of the development process allows the placement of buffers to be delayed until later during the placement process, when layout information is more definite and precise. The above description of the preferred embodiment of the present invention has been given for purposes of illustration only, and variations thereof will be readily apparent to those skilled in the art. For example, although Verilog has been used as the preferred language for initial input of the circuit under development, other appropriate hardware description languages may of course be used. Also, although implementation of the data model using object-oriented C++ techniques has been disclosed, other programming languages and paradigms may also be workable. Similarly, alternative object hierarchies

may be used. Such variations fall within the scope of the present invention. Thus, the scope of the present invention should be limited only by the appended claims.

What is claimed is:

1. A common data model representing a circuit that will be fabricated on an integrated circuit chip comprising:

a data representation including a plurality of objects that together represent the circuit, certain ones of the objects including a netlist portion that represents a corresponding portion of the circuit, and each of the objects:

being logically correlated to at least one other object so that all of the objects describe the circuit; and

each of the objects, once associated with a physical location is adapted for subsequent retrieval using an area query corresponding to the physical location.

2. The model according to claim **1** wherein the physical location association of objects is implemented using hierarchical partitioning.

3. The model according to claim **2** wherein the bierarchical partitioning is implemented using a tree.

4. The model according to claim **3** wherein the circuit is represented within an area, with a plurality of cutlines that partition the area into a plurality of rectagles.

5. The model according to claim **4** wherein the tree contains a plurality of leaf nodes, and each of the leaf nodes corresponds to one of the cutlines.

6. The model according to claim **5** wherein the tree includes a linked list that identifies each cell that lies on a particular one of the cutlines.

7. The model according to claim **5** wherein the tree contains a plurality of non-leaf nodes, each of the non-leaf nodes associated with one of the leaf nodes, and each of the non-leaf nodes, containing at least two child nodes, each child node corresponding to an area on an opposite side of the cutline associated with the one leaf node.

8. The model according to claim **3** wherein certain of the objects represent cells.

9. The model according to claim **3** wherein certain of the objects represent a net or a part of a net.

10. The model according to claim **3** wherein certain of the objects represent pins.

11. The model according to claim **1** wherein the each of the objects corresponding to each of the physical locations is maintained in an active memory.

12. The model according to claim **11** wherein the subsequent retrieval of objects corresponding to the physical location of the area query causes the retrieval of all objects associated with the physical location to be retrieved into an active memory.

13. The model according to claim **12** wherein the retrieval of all objects associated with the physical location is from a disk storage.

14. The model according to claim **1** wherein the model allows for insertion of cutlines to frame a particular area of interest.

15. The model according to claim **1** wherein the area query takes place either immediately after synthesis or during final placement and routing.

16. The model according to claim **1** wherein the model is configured to allow for a logical query to take place.

17. The model according to claim **16** wherein the logical query of one object provides at least another object that is logically related to the one object.

* * * * *

B

US006519745B1

(12) **United States Patent**
Srinivas et al.

(10) **Patent No.: US 6,519,745 B1**
(45) **Date of Patent: Feb. 11, 2003**

(54) **SYSTEM AND METHOD FOR ESTIMATING CAPACITANCE OF WIRES BASED ON CONGESTION INFORMATION**

(75) Inventors: **Prasanna Venkat Srinivas**, Cupertino, CA (US); **Manjit Borah**, Cupertino, CA (US); **Premal Buch**, Cupertino, CA (US)

(73) Assignee: **Magma Design Automation, Inc.**, Cupertino, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/579,966**

(22) Filed: **May 26, 2000**

(51) **Int. Cl.**[7] **G06F 9/45**; H03K 19/173; H03K 19/177; G01R 31/26

(52) **U.S. Cl.** **716/5**; 326/38; 326/39; 438/17; 716/2; 716/11; 716/13; 716/6

(58) **Field of Search** 702/81; 716/1–21; 257/277; 326/36, 39; 327/99; 438/17

(56) **References Cited**

U.S. PATENT DOCUMENTS

6,185,722 B1 * 2/2001 Darden et al. 716/5
6,311,139 B1 * 10/2001 Kuroda et al. 702/81
6,327,693 B1 * 12/2001 Cheng et al. 716/2

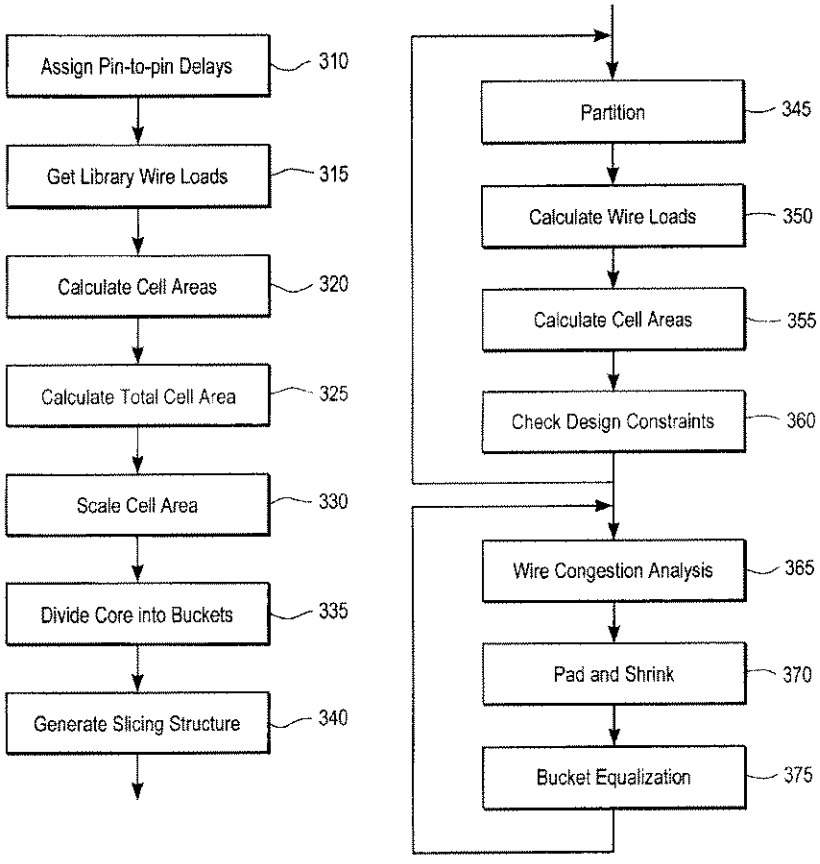
* cited by examiner

Primary Examiner—Vuthe Siek
Assistant Examiner—Helen Rossoshek
(74) *Attorney, Agent, or Firm*—Pillsbury Winthrop LLP

(57) **ABSTRACT**

A system for calculating interconnect wire lateral capacitances in an automated integrated circuit design system subdivides the chip area of a circuit design to be placed and routed into a coarse grid of buckets. An estimate of congestion in each bucket is computed from an estimated amount of routing space available in the bucket and estimated consumption of routing resources by a global router. This congestion score is then used to determine the spacing of the wires in the bucket which is in turn used to estimate the capacitance of the wire segment in the bucket.

8 Claims, 7 Drawing Sheets



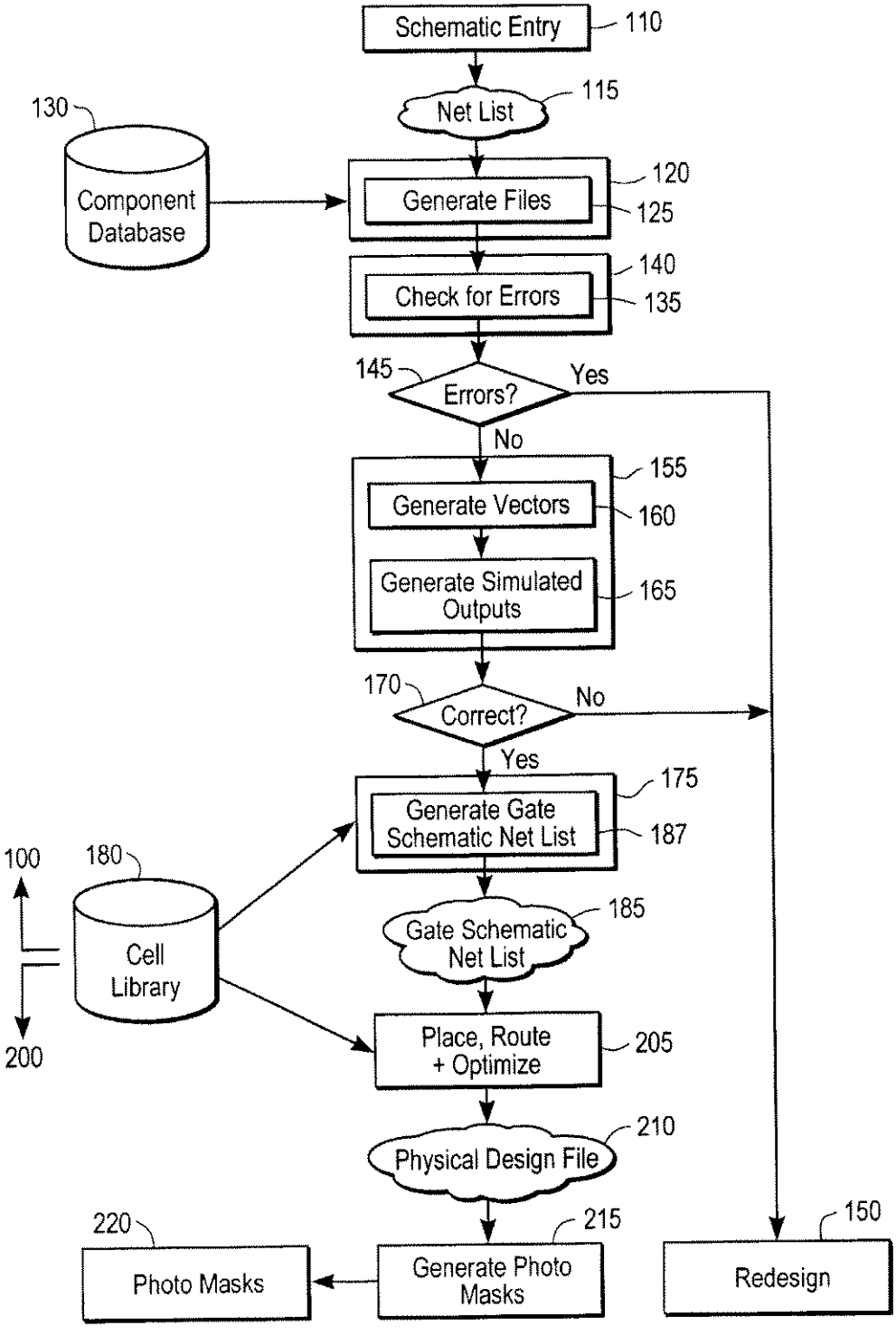

Schematic Entry
110
Net List
115
130
Component Database
120
Generate Files
125
140
Check for Errors
135
145
Errors?
Yes
No
155
Generate Vectors
160
Generate Simulated Outputs
165
170
Correct?
No
Yes
175
Generate Gate Schematic Net List
187
100
180
Cell Library
200
185
Gate Schematic Net List
Place, Route + Optimize
205
210
Physical Design File
220
Photo Masks
215
Generate Photo Masks
150
Redesign

FIG. 1

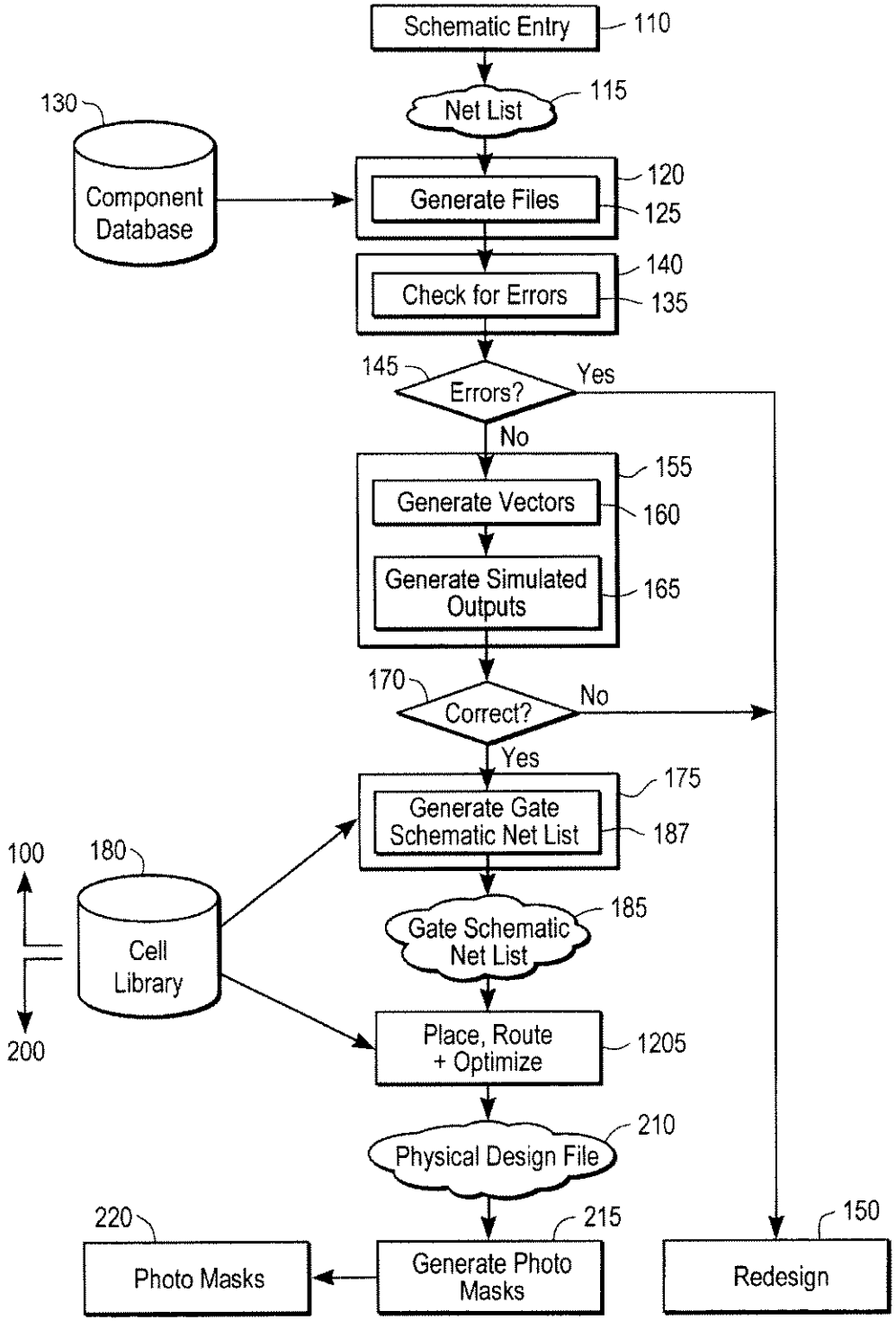

Schematic Entry
110
Net List
115
130
Component Database
Generate Files
120
125
Check for Errors
140
135
145
Errors?
Yes
No
Generate Vectors
155
160
Generate Simulated Outputs
165
170
Correct?
No
Yes
Generate Gate Schematic Net List
175
187
100
180
Cell Library
200
Gate Schematic Net List
185
Place, Route + Optimize
1205
Physical Design File
210
220
Photo Masks
Generate Photo Masks
215
Redesign
150

FIG. 2

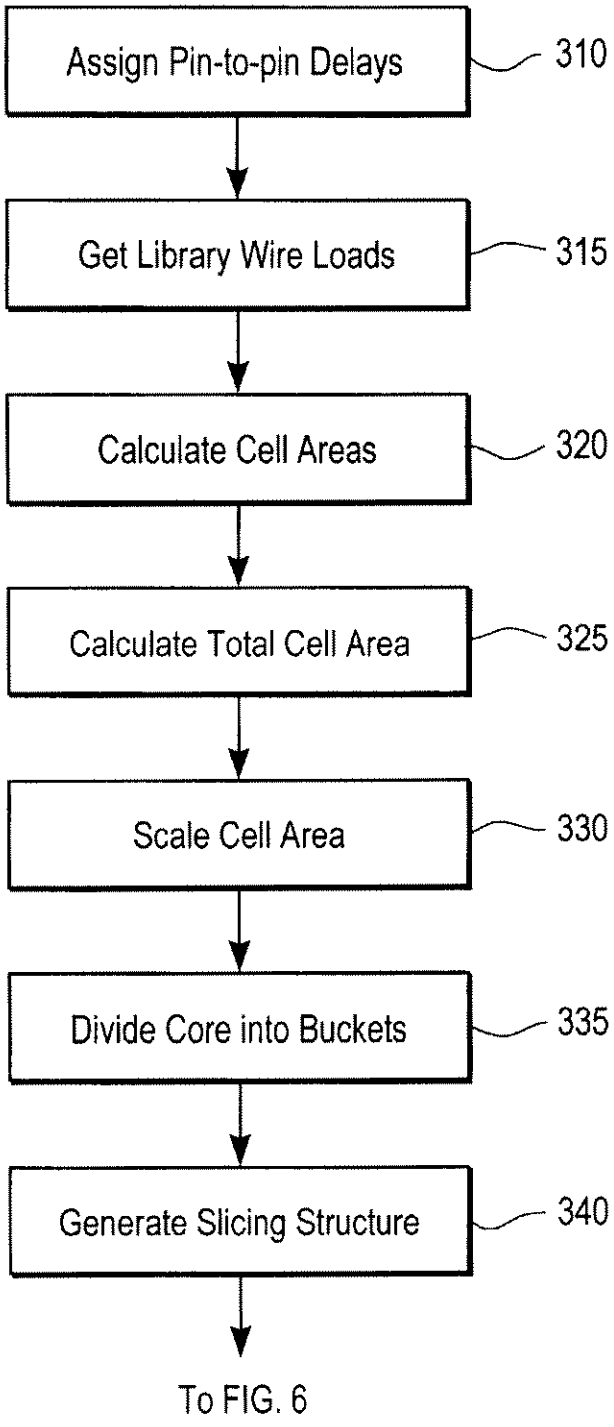

Assign Pin-to-pin Delays
310
Get Library Wire Loads
315
Calculate Cell Areas
320
Calculate Total Cell Area
325
Scale Cell Area
330
Divide Core into Buckets
335
Generate Slicing Structure
340
To FIG. 6

FIG. 3

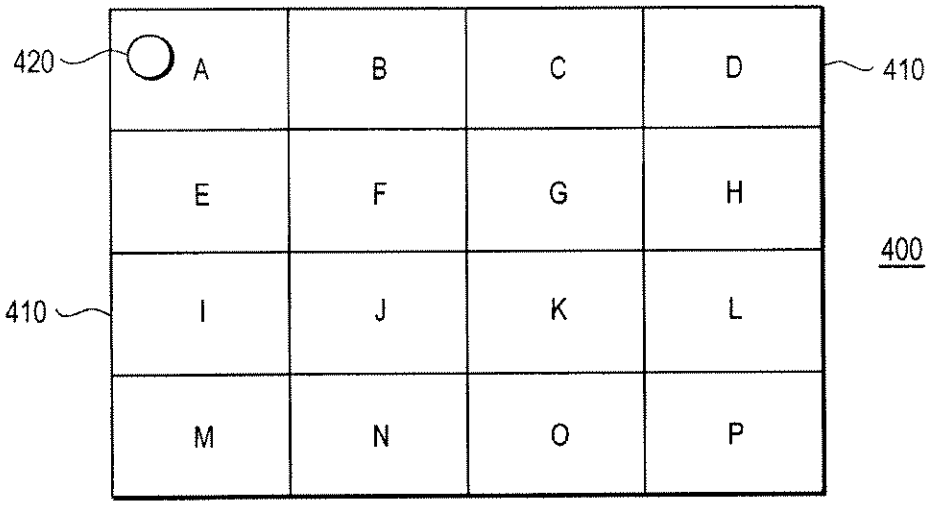

420
A
B
C
D
410
E
F
G
H
400
410
I
J
K
L
M
N
O
P

FIG. 4

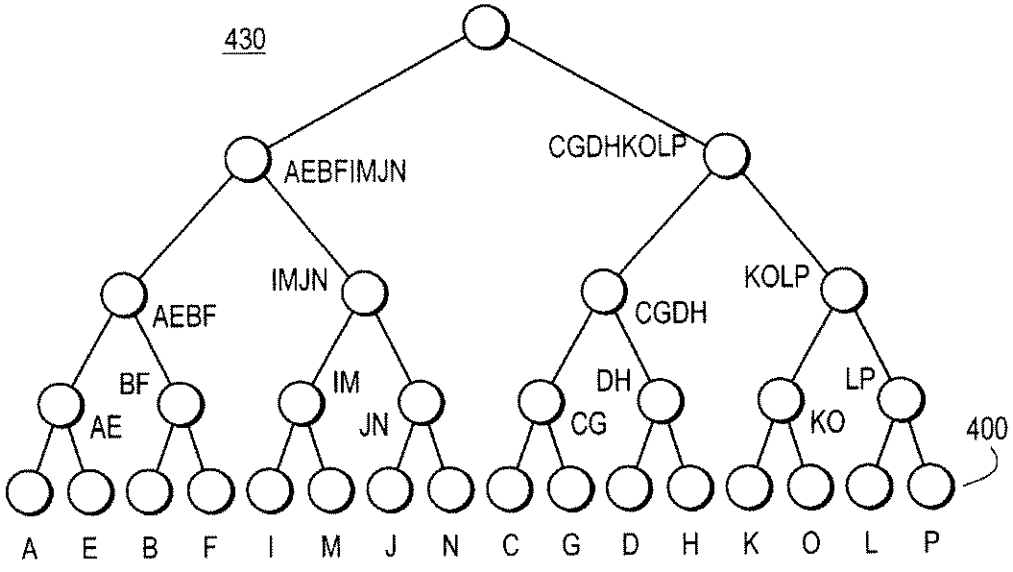

430
AEBFIMJN
CGDHKOLP
AEBF
IMJN
CGDH
KOLP
AE
BF
IM
JN
CG
DH
KO
LP
400
A E B F I M J N C G D H K O L P

FIG. 5

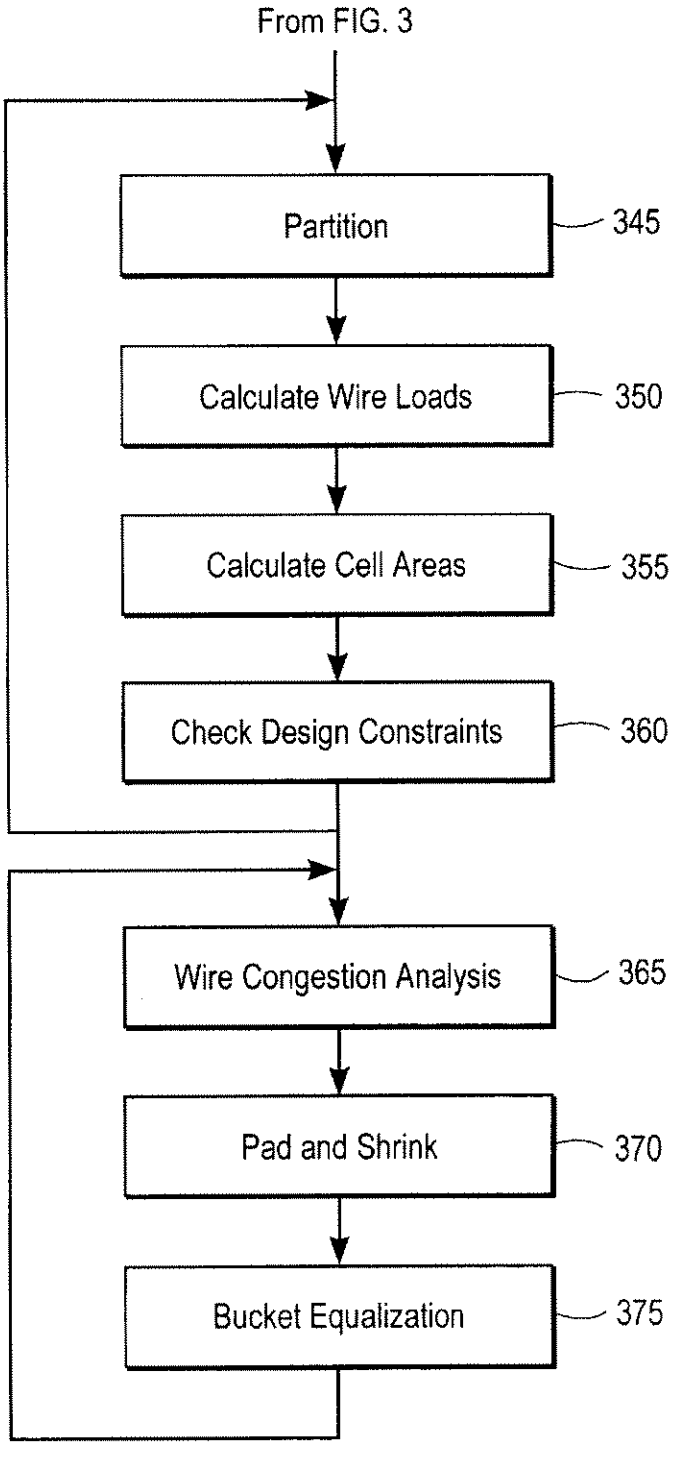

From FIG. 3
Partition
345
Calculate Wire Loads
350
Calculate Cell Areas
355
Check Design Constraints
360
Wire Congestion Analysis
365
Pad and Shrink
370
Bucket Equalization
375

FIG. 6

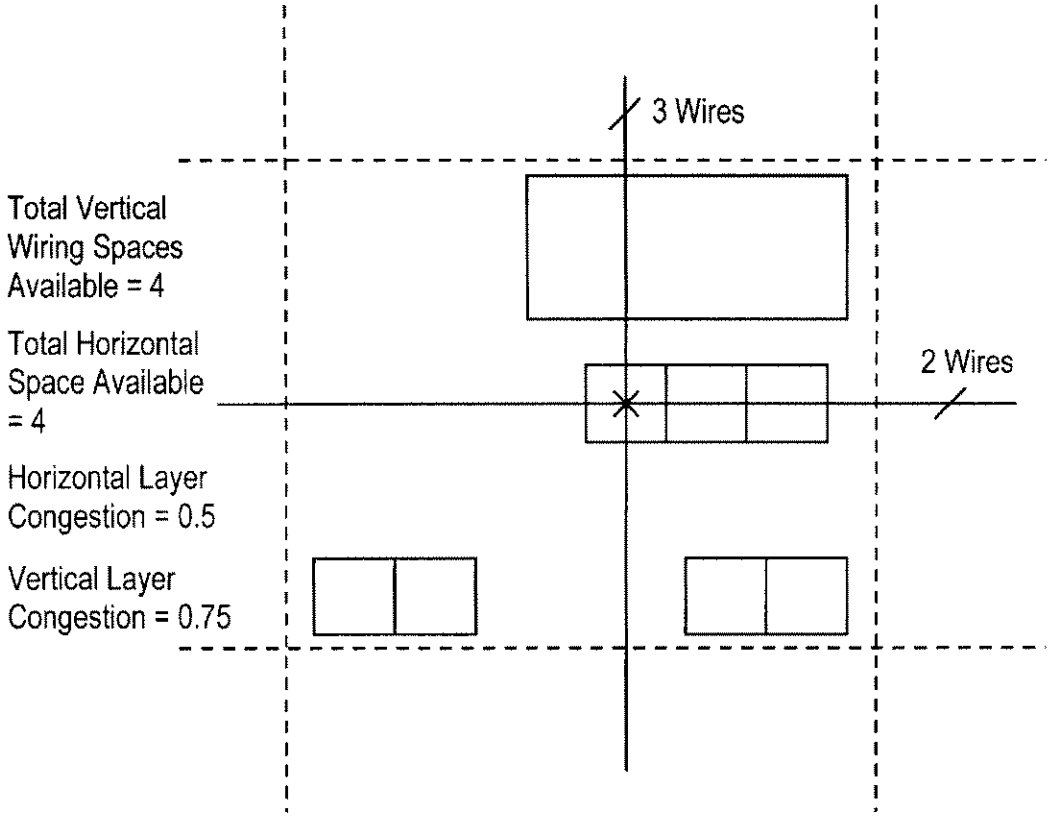

3 Wires
Total Vertical Wiring Spaces Available = 4
Total Horizontal Space Available = 4
2 Wires
Horizontal Layer Congestion = 0.5
Vertical Layer Congestion = 0.75

FIG. 7

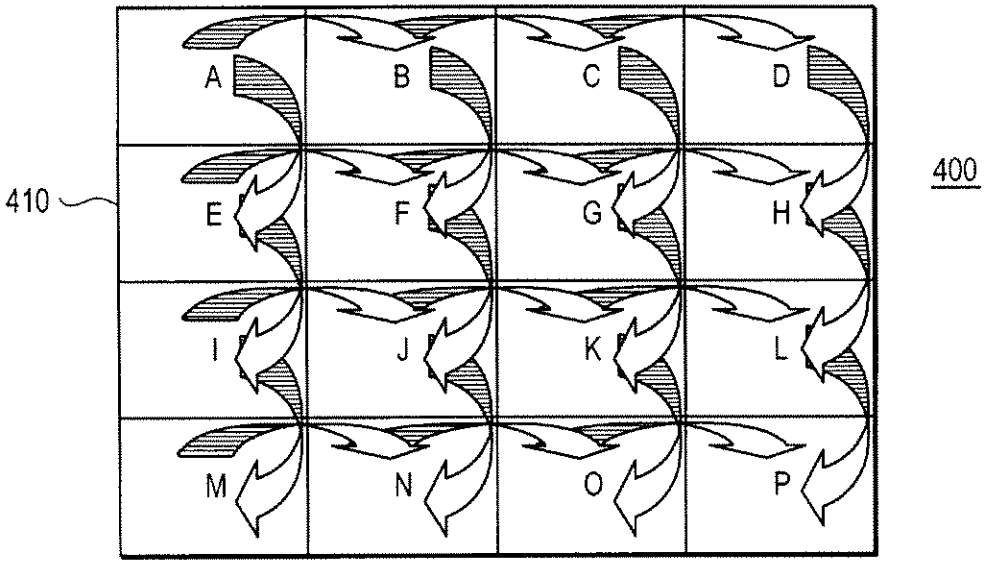

A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
410
400

FIG. 8

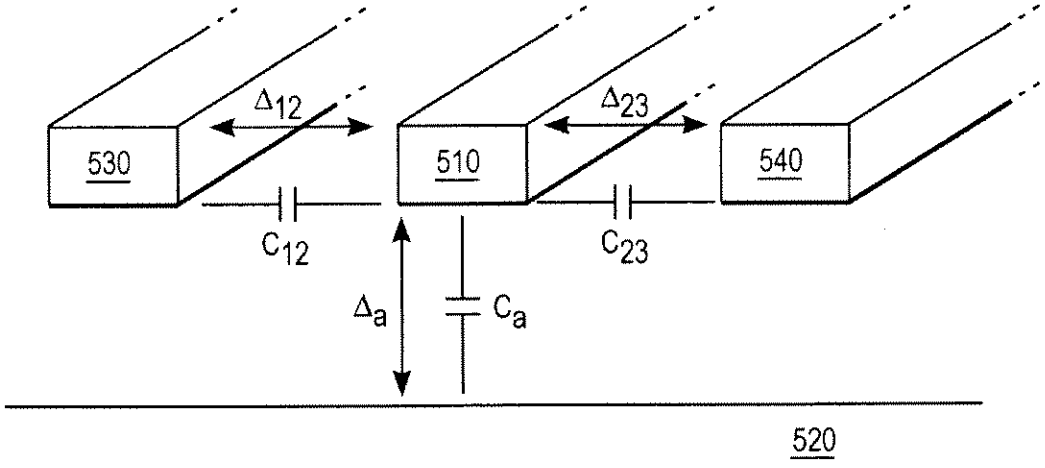

530
510
540
Δ_{12}
Δ_{23}
C_{12}
C_{23}
Δ_a
C_a
520

FIG. 9

SYSTEM AND METHOD FOR ESTIMATING CAPACITANCE OF WIRES BASED ON CONGESTION INFORMATION

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention is directed to digital logic design systems. More particularly, the invention is directed to automated digital logic synthesis and placement systems for integrated circuits, and to performance optimization of digital integrated circuits.

2. Background of the Related Art

Prior art computer aided design (CAD) systems for the design of integrated circuits (ICs) and the like assist in the design thereof by providing a user with a set of software tools running on a digital computer. In the prior art, the process of designing an integrated circuit on a typical CAD system is done in several discrete steps using different software tools.

The design process can be broadly divided into two phases. The initial phase **100** (shown in FIG. 1) of selecting the right components and connecting them so that the desired functionality is achieved is called logical synthesis. The second phase **200**, in which the selected components are placed within the confines of the chip boundaries and the connecting wires are laid out in order to generate the photographic masks for manufacturing, is called physical synthesis.

First, in the logical synthesis phase **100** a schematic diagram of the integrated circuit is entered interactively in Step **110** to produce a digital representation **115** of the integrated circuit elements and their interconnections. This representation **115** may initially be in a hardware description language such as Verilog or VHDL and then translated into a register transfer level (RTL) description in terms of predesigned functional blocks, such as memories and registers. This may take the form of a data structure called a net list.

Next, a logic compiler **120** receives the net list in Step **125** and, using a component database **130**, puts all of the information necessary for layout, verification and simulation into object files whose formats are optimized specifically for those functions.

Afterwards, in Step **135** a logic verifier **140** preferably checks the schematic for design errors, such as multiple outputs connected together, overloaded signal paths, etc., and generates error indications in Step **145** if any such design problems exist. In many cases, the IC designer improperly connected or improperly placed a physical item within one or more cells. In this case, these errors are flagged to enable her to correct the layout cells in Step **150** so that they perform their proper logical operation.

Also, in Step **135** the verification process preferably checks the cells laid out by hand to determine if multiple design rules have been observed. Design rules may include the timing requirements of the circuit, the area occupied by the final design and parameters derived from other rules dictated by the underlying manufacturing technology. These design rules are provided to integrated circuit designers to ensure that a part can be manufactured with a high degree of yield. Most design rules include hundreds of parameters and, for example, include pitch between metal lines, spacing between diffusion regions in the substrate, sizes of conductive regions to ensure proper contacting without electrical short circuiting, minimum widths of conductive regions, pad

sizes, and the like. If a design rules violation is identified in Step **150**, this violation is preferably flagged to the IC designer so that she can properly correct the cells so that they are in accordance with the design rules in Step **150**.

Then, using a simulator **155** the user of the CAD system may prepare a list of vectors representing real input values to be applied to a simulation model of the integrated circuit in Step **160**. This representation may be translated into a form which is better suited to simulation. This representation of the integrated circuit is then operated upon by the simulator which produces numerical outputs analogous to the response of a real circuit with the same inputs applied in Step **165**. By viewing the simulation results, the user may then determine in Step **170** if the represented circuit will perform correctly when it is constructed. If not, she may re-edit the schematic of the integrated circuit, re-compile it and re-simulate it in Step **150**. This process is performed iteratively until the user is satisfied that the design of the integrated circuit is correct.

Then, the human IC designer may present as input to a logic synthesis tool **175** a cell library **180** and a behavioral circuit model. The behavioral circuit model is typically a file in memory which looks very similar to a computer program, and the model contains instructions which logically define the operation of the integrated circuit. The logic synthesis tool **175** maps the instructions from the behavioral circuit model to one or more logic cells from the library **180** to transform the behavioral circuit model to a gate schematic net list **185** of interconnected cells in Step **187**. The gate schematic net list **185** is a database having interconnected logic cells which perform a logical function in accordance with the behavioral circuit model instructions. Once the gate schematic net list **185** is formed, it is provided to a place and route tool **205** to begin the second phase of the design process, physical synthesis.

The place and route tool **205** is preferably then used to access the gate schematic net list **185** and the library cells **180** to position the cells of the gate schematic net list **185** in a two-dimensional format within a surface area of an integrated circuit die perimeter. The output of the place and route step may be a two-dimensional physical design file **210** which indicates the layout interconnection and two-dimensional IC physical arrangements of all gates/cells within the gate schematic net list **185**. From this, in Step **215** the design automation software can create a set of photographic masks **220** to be used in the manufacture of the IC.

One common goal in chip design involves timing performance. The timing performance of the chip is determined by the time required for signals to propagate from one register to another. Clock signals driven at a certain frequency control storage of data in the registers. The time required for a signal to propagate from one register to another depends on the number of levels of cells through which the signal has to propagate, the delay through each of the cells and the delay through the wires connecting these cells. The logic synthesis phase **100** influences the number of levels and the propagation delay through each cell because in it the appropriate components are selected, while the physical synthesis **200** phase affects the propagation delay through the wires.

During the process of timing optimization during physical design in Step **205**, circuit timing is evaluated based on an initial placement and selection of cell strengths. The feedback from the timing analysis is used to drive repeated improvements to the placement software and the selection of the strengths of the cells. The automation software may also perform buffering on some parts of the circuit to optimize

the timing performance by inserting repeater cells, i.e., buffers, to speed up certain paths. Preferably, the optimization software tentatively applies one such modification, evaluates the timing and other constraints (such as design rules dictating capacitance limits) to determine if the step is acceptable and then makes the change permanent if it is deemed acceptable.

The interconnection of the cells in the placing and routing of Step **205** generally involves interconnect wiring having between two and seven metal layers. The delay through an interconnect wire depends on the capacitance of the wire, its resistance and, to a lesser extent, the inductance of the wires. The capacitance of a wire **510** (see FIG. **9**) consists mainly of the capacitance C_a due to the overlap of the wire with the layer **520** above or below, called the area capacitance C_a, and the capacitance due to the overlap along the side walls with other signal wires **530** and **540** adjacent to it, called the lateral capacitance $C_L=C_{12}+C_{13}$. The capacitance of a given wire such as wire **510** can be calculated on a case-by-case basis as is known in the art, and will primarily depend on the wire dimensions D_W and D_T as well as the distance D_L of the wire **510** from the other layers **520** and the distance D_{12} and D_{23} of the wire **510** from the other wires **530** and **540**. In deep sub-micron manufacturing technologies the widths of the wires are becoming thinner and thinner, making them tall and narrow. As a result, under current development trends the lateral capacitance C_L is becoming a dominant component of the total wire capacitance.

During the process of timing optimization during physical design, circuit timing is evaluated based on an initial placement and selection of cell strengths. The feedback from the timing analysis is used to drive the repeated improvements to the placement software and the selection of the strengths of the cells. Since physically laying out all the wires without violating design rules and maintaining good delays is a very time consuming step, a Steiner tree-based topology is used to estimate the area and delay due to the wires based on the current cell placement.

An Elmore delay model is commonly used to compute the wire delay based on the Steiner tree-based topology. However, computing the capacitance of the wires based on the Steiner tree topology is difficult, because nothing is known about the spacing of the wires and the adjacency of different signals at this point in design. Existing approaches use the worst case scenario, and assume that there are wires adjacent to the signal wire in consideration and thus tend to over-estimate the capacitance. Since the optimization software depends on the feedback from the timing analysis, accurate estimation of the capacitance is crucial to the success of the optimizations.

SUMMARY OF THE INVENTION

In view of the above problems of the prior art, it is an object of the present invention to provide a method of estimating the effect of adjacent wires on the capacitance of a signal wire in a circuit design which provides estimates superior to prior art techniques.

It is another object of the present invention to provide a method of estimating the effect of adjacent wires on the capacitance of a signal wire in a circuit design which provides estimates for better than worst case scenarios.

One of the indicators for the expected spacing of the wires in the final routed circuit is the density of the interconnect wires at a given point, i.e., the wires' congestion. The greater the number of wires that passes through a given bucket, the greater the density of wires in that bucket will be and as a result the spacing tends to be smaller to fit all the wires. It would be advantageous to make use of the congestion in a circuit to derive an early estimate of the capacitance.

Thus, it is yet another object of the present invention to provide a method of estimating the effect of adjacent wires on the capacitance of a signal wire in a circuit design which provides estimates based on an estimate of congestion in the design.

The above objects are achieved according to an aspect of the invention by subdividing the chip area of a circuit design to be placed and routed into a coarse grid of buckets. An estimate of congestion in each bucket is computed from an estimated amount of routing space available in the bucket and estimated consumption of routing resources by a global router. This congestion score is then used to determine the spacing of the wires in the bucket which is in turn used to estimate the capacitance of the wire segment in the bucket.

BRIEF DESCRIPTION OF THE DRAWINGS

These and other objects, features and advantages of the present invention are better understood by reading the following detailed description of the preferred embodiment, taken in conjunction with the accompanying drawings, in which:

FIG. **1** is a flowchart of an integrated circuit design process according to the prior art;

FIG. **2** is a flowchart of an integrated circuit design process according to a preferred embodiment of the present invention;

FIGS. **3** and **6** are a flowchart showing a place and route process according to the preferred embodiment;

FIG. **4** shows a coarse grid with buckets in a circuit used with the preferred embodiment;

FIG. **5** shows a slicing structure used in the preferred embodiment;

FIG. **7** shows a cell upon which congestion calculations are performed according to the preferred embodiment;

FIG. **8** shows a bucket mincut process according to the preferred embodiment; and

FIG. **9** shows the major components in routing wire capacitance.

DETAILED DESCRIPTION OF PRESENTLY PREFERRED EXEMPLARY EMBODIMENTS

In one model, the delay through a single logic gate can be represented as

$$d=g\cdot h+p \tag{1}$$

where d is the delay, g is a parameter called the "logical effort" of the gate, h is a parameter called the "electrical effort" of the gate, and p is the parasitic or fixed part of the delay g, in turn, is defined by

$$g_{gate}=\frac{R_{gate\_min}C_{Gate\_min}}{R_{inv\_min}C_{inv\_min}} \tag{2}$$

where gate_min refers to a minimum-sized gate and inv_min to a minimum-sized inverter. h, in turn, is defined by

$$h = \frac{c_{out}}{c_{in}} \quad (3)$$

where c_{out} is the capacitance out of the gate and c_{in} is the capacitance into the gate.

In a constant delay approach to cell placement, the pin-to-pin delay of each cell is fixed early on in the optimization flow. This delay is maintained independently of the load a cell drives. In order to keep delay constant, the size of the cell is adjusted according to the load that it drives. As a result, the area of each cell in the design varies with the load that it drives. The area of each cell is

$$a=b+s\cdot c_{out} \quad (4)$$

where b and s are constants related to the logic of the cell and the chosen constant delay for the cell. Thus, the total area of the netlist is, in matrix notation for plural gates,

$$A=B+SC \quad (5)$$

Approximating the input load at each pin of the cell by c_k/h_k, the total load at the output of a cell i is

$$c_i = \frac{c_j}{g_j} + \frac{c_k}{g_k} \ldots + d_i \quad (6)$$

or, alternatively,

$$c_i = \sum_{\text{fanout}} \frac{c_k}{h_k} + d_i \quad (7)$$

where d_i is the wire load. That is, the total load at the output of cell i is the sum of all its fanout loads plus the load of the wire connecting the cell to its fanouts. In matrix notation for plural gates,

$$C=HC+D \quad (8)$$

$$(I-H)C=D \quad (9)$$

Setting G=I−H,

$$GC=D \quad (10)$$

$$C=G^{-1}D \quad (11)$$

where C is the output capacitance of all gates in the circuit and D is the wire load. Thus, according to the last equation above, the output load of the cells in the circuit can be found once the placement is known. Then, the size of each cell can be found to keep its delay constant. The area of each cell in the netlist denoted by A is

$$A=K_1+K_2{}^TC \quad (12)$$

and substituting Equation (10) gives

$$A=K_1+K_2{}^TG^{-1}D \quad (13)$$

Since the load of each wire can be represented as d=u·l, where d is the wire load, u is the load per unit length of wire and l is the total length of the wire,

$$A=K_1+\mu K_2{}^TG^{-1}l. \quad (14)$$

and combining constant terms,

$$A=K_1+WL \quad (15)$$

Thus, in order to minimize the circuit area one can minimize WL, where the matrix W may be viewed as a set of weights of the wire lengths L.

Generally, the cell is modeled as a rectangle, with the height of the rectangle being the height of a standard cell row. Thus, the width and therefore the area of the cell are functions of the load.

The preferred embodiment of the present invention processes a data structure representative of the circuit being placed and routed. Preferably, this is done on a digital computer as is known in the art. The data structure may be a netlist or other suitable structure known in the art; however, it is preferably a data model of the type disclosed in the U.S. patent application Ser. No. 09/300,540 to Van Ginneken et al; however, other simpler structures may be used as well.

The overall flow of a place and route process **1205** (see FIG. **2**) according to a preferred embodiment of the present invention is shown in FIGS. **3** and **6**. Since a design constraint of the placement process is that the delay across a net be constant, in the preferred embodiment the area of a cell is dependent on the load it drives. In turn, the load of a wire is not known with certainty until the placement process is finished. Thus, to make an initial placement of cells within the core some initial estimations are preferably made. Each cell is assigned a pin-to-pin constant delay in Step **310**. Appropriate techniques will be readily apparent to those skilled in the art; however, pin delay assignment is preferably done according to the technique described in the U.S. patent application Ser. No. 09/300,666. Throughout the placement process, this delay will be maintained constant and the area of the cell varied according to the load it drives in order to achieve the assigned delay.

To make the initial cell placement, the area of each cell is calculated in Step **320** using wire loads obtained from the cell library in Step **315** and substituted into Equation 15. Although cells of varying power levels are available only in discrete steps in the target cell library, this phase of the technique proceeds as if a continuous spectrum of cell power levels are available and selects a cell from the library closest to the size ultimately selected as one of the final steps of the process.

The total cell area A_{total} is determined by adding up the areas of all the cells in Step **325**, and the sizes of the cells are scaled to achieve a target percentage of core utilization in Step **330**. Based on this, standard cell rows are created.

More specifically, the core **400** where the cells are placed is divided into coarse placement regions called buckets **410** as shown in FIG. **4**. Each bucket **410** is a small rectangular region within the core **400**. Buckets **410** have equal dimensions but the placeable area within a bucket **410** depends on the presence of blockages such as macros in the bucket **410**. A bucket **410** can accommodate about fifty average-sized standard cells **420**. Then, in Step **340** a slicing structure or binary tree **430** is built whose leaves **440** are the coarse buckets **410**. For example, a core **400** having a 4×4 matrix of buckets **410** imposed thereon (of course, in practice there will be a much greater number of buckets **410**) as shown in FIG. **4** can be represented by the slicing structure **430** shown in FIG. **5**.

Cells **420** are assigned to the buckets **410** so that the total area of cells **420** within each bucket **410** closely matches the area of that bucket **410**. This is done by an iterative bipartitioning of the data model. First, a horizontal or vertical cut of the core **400** is chosen. The total area available on each side of the partition is computed. Cells **420** are divided using quadratic placement (see below) and a mincut technique

(see, e.g., Fiduccia et al., "A Linear Time Heuristic for Improving Network Partitions", ACM/IEEE Design Automation Conference, 1982, pp. 175–81) on each side so that total wire length is minimized. This iteration continues until a desired resolution, e.g., a bucket **410**, is reached.

Later, each cell **420** is assigned to one of the buckets **410** using a partitioning technique in Step **345** as shown in FIG. **6**. The second placement is done instead of a single cell-level placement because the first placement is done with a coarse grid, i.e., buckets containing hundreds of cells. Here, the cells are placed into their corresponding buckets. In the later stage when other optimizations such as choosing the size and adding repeaters are done and the netlist is more stable, the second, detailed placement is done. This placement step places a cell in its actual location. In this stage, some cells may move from their originally-assigned bucket to a neighboring bucket to make room for the cells in a highly populated bucket in order to reduce congestion.

A good placement of cells **420** is one that can be easily routed and satisfies the given timing constraints for the logic circuit. Quadratic placement, and in particular Gordian quadratic placement, finds a legal placement while minimizing the total squared wire length in the circuit and is the placement technique preferably used. Gordian quadratic placement is well-known in the art as shown by, e.g., Klienhans et al., "GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization", IEEE Trans on Computer-Aided Design, v. 10, n. 3 (Mar. 1991), pp. 356–365, and for simplicity will be generally described below.

The problem is independently solved for the x and y coordinates. Briefly describing the process for the x coordinates (the process for the y coordinates is similar), quadratic placement solves the following equation subject to a constraint Hx=t (to account for physical realities such as overlapping cells and the like) to minimize total wire length during placement:

$$\tfrac{1}{2}(\Sigma c_{ij}(x_i-x_j)^2+\Sigma c_{ij}(x_i-b_j)^2) \quad (16)$$

$$\tfrac{1}{2}x^tAx-x^td+\text{constant} \quad (17)$$

x is the location of cells **420** and star nets. Star nets are nets with more than fifteen pins. A star net is treated like a cell **420**. All cells **420** attached to a star net are considered to be attached to the center of the net through a two-pin net. Star nets are used to reduce the number of fill-ins in the matrix A. The weight of a net k_i is 2/(number of pins). The weight of a net connecting a cell **420** to the center of a star net is 1.b has the locations of fixed points. Fixed points are pins of pads or macros. The diagonal elements of A are non-zero and are computed as follows:

$$a_{ij}=\text{SUM}k_i \quad (18)$$

Any cell **420** connecting to cell ii through a non-star net and a star net connecting to a cell i contribute to the summation. The element a_{ij} is non-zero if cells i and j are connected through a net.

$$a_{ij}=-\text{SUM } k_i \quad (19)$$

The contribution comes from the nets connecting cells i and j.

$$d_i=\text{SUM } b_jk_i \quad (20)$$

The contribution comes from all constant pins attached to cell i. The x coordinates for a placement that minimizes the total wire length is obtained by solving

$$Ax=d \quad (21)$$

The initial constraints for quadratic placement assumes the center of mass for all cells **420** on the chip is the center of the chip. If the area of each cell is a_i, $\Sigma a_ix_i=X_{center}$ forms the first constraints for quadratic placement.

The place and route software then performs global routing on the placed cells **420**. Global routing is a coarse level routing that uses Steiner tree-based topologies to connect the bucket centers. The use of Steiner trees in such contexts is well-known in the art; see, for example, Borah et al., "An Edge-Based Heuristic for Steiner Routing", IEEE Transactions on Circuits and Systems 1563–68 (1993) and Griffith et al., "Closing the Gap: Near-Optimal Steiner Trees in Polynomial Time", IEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, v. 13,n. 11 (Nov. 1994) 1351–65 (both incorporated herein by reference) and will not be described in greater detail herein.

When the global routing software adds a wire through a bucket (either during the initial Steiner tree calculation or during recalculations as described below), the budget for the appropriate layer is adjusted to reflect the use of one routing resource from the bucket. More specifically, in order to concisely describe congestion in a bucket, the available routing space in each routing layer is estimated in advance by the computer and assigned to the bucket as a budget for that layer. Every time a wire is routed through the bucket in a particular routing layer, the budget for that layer is adjusted to reflect that one routing space has been spent. The lower the number of available spaces, the greater the congestion in the bucket in the given routing layer.

FIG. **7** illustrates an example of a bucket with three routing wires running through it in the vertical direction in one layer and two routing wires running through it in the horizontal direction in another layer. The congestion score for a bucket is defined as the ratio of the routing resources used so far to the total routing resources available in the bucket. For example, the vertical layer of the bucket of FIG. **7** has a total of four routing spaces (its actual dimensions will be larger than this due to design rule considerations as described in greater detail below), and three vertical wires are going through the bucket. Thus, this bucket has a congestion score of 0.75 along the layer in the vertical routing direction. Similarly, the horizontal routing direction has 2 wires going through it, resulting in a congestion score of 0.5.

When the capacitance for a wire is estimated, e.g., during optimization operations such as inserting repeaters, choosing appropriate strengths, etc., the congestion score for each of the buckets the wire goes through is used to compute the spacing for the wire segment in the bucket. To calculate the congestion score for a bucket, it is important to remember that design rules require a certain amount of free space on each side of a routing wire. Thus, a layer having spaces to accommodate ten routing wires which has five routed therethrough (wire-space-wire-space . . .) may have 100% congestion because the layer cannot accommodate any more routine wires while maintaining the separation design rule.

Given the above, one can see that a layer having 50% congestion can roughly be thought of as having a number of routing wires with one empty space on one side and two empty spaces on the other—omitting the required space on each side of the wire, it has one wire for two usable spaces). A layer having 33% congestion can similarly be thought of as having routing wires with one space on one side and three spaces on the other, or with two spaces on one side and two spaces on the other (one wire in three usable spaces). A layer having 25% congestion can be thought of as having routing

wires with two spaces on one side and three spaces on the other (one wire in four usable spaces). Further discrete points can be derived with more sparse routing layers.

Given a maximum of three spaces between adjacent wires in a routing layer, a suitable equivalence chart might be:

Congestion Score	Corresponds to
0–23%	3 spaces/3 spaces
23–29%	2 spaces/3 spaces
29–42%	2 spaces/2 spaces or 1 space/3 spaces
42–75%	1 space/2 spaces
75–100%	1 space/1 space

In a variation on the preferred embodiment the spacing value for the wires are rounded to the nearest integers and a congestion score of more than 75% is considered to have a spacing of 1 on both sides (the minimum spacing which could conform to the design rules), a congestion score of 50% represents a spacing of 1 on one side and 2 on the other side and a score below 50% uses spacing of 2 on both sides. Hence if the initial number of routable spaces available in a bucket is 10 and we have routed three wires through it then the congestion score for the bucket is 30%. Each wire running through this bucket will be assumed to have a spacing of 2 on both sides.

The interconnect wire capacitance determination makes use of the spacing thus computed to compute the lateral capacitance for the wire segment. As is known in the art, this can be done by taking a number of parameters of the technology into account; for example, design rule spacings, distance of the routing layer from the substrate, thickness and material of the wires and the like. Generally speaking, the most important factors are the width and spacing of the wires.

The capacitance of the whole net is then computed by adding up the contribution of each wire segment belonging to the net. It may then be used where needed in optimization operations, e.g., inserting repeaters, choosing appropriate strengths, etc.

Based on the routed cell arrangement resulting from the initial Steiner tree arrangement, the cell areas can be recalculated in Step **355** using Equation 15 with the new wire loads substituted therein. At this point, the cell placement will likely be somewhat unbalanced. This imbalance may take several forms:

- widely varying cell utilization percentages—for example, if the core utilization before the cell area recalculation is 90%, after recalculation some buckets **410** will have higher utilization percentages and some buckets **410** will have lower utilization percentages. This is undesirable because, for example, overutilized buckets **410** may present obstacles to wire routing or usage of pads.
- cell recalculation enlarges the size of some cells **420** so that they do not fit within their buckets **410**, or so that they overlap other cells **420**.
- cell recalculation results in too much wasted area, i.e., unutilized core area.

To correct these problems, an iterative procedure is used. First, the current layout is checked to see if it meets given utilization constraints such as core utilization percentage in Step **360**. If so, the placement procedure is complete and this part of the routine ends. If not, i.e., if the total area A_{total} of the cells **420** does not fit in the core **400** within the given predetermined utilization constraints, the procedure returns to Step **345** where repartitioning is conducted by coarse placement based on the last-determined cell areas from Step **350**, and the repartition-recalculation-checking loop is iteratively executed again based on the newly-calculated cell areas and wire loads to further converge toward an acceptable placement.

Additional analysis shows that it is always possible to find a floorplan where the total area of the cells **420** matches the core area. Consider a coarse placement where each cell **420** has a location (xi, yi) and an area based on the load it drives as outlined above. From Equation 14 above, the total area of the design is

$$A_{Cell}=K_1+WL \tag{22}$$

Now, assume both x and y directions are stretched by a factor of α. The length of each wire is increased by α, and since the cell area is linearly dependent on the wire length,

$$A_{Scaled\ Cell}=\alpha(K_1+WL) \tag{23}$$

However, by scaling the core **400** by a factor of a its area will increase quadratically:

$$A_{Scaled\ Core}=\alpha^2 A_{Core} \tag{24}$$

Since the core area increases more rapidly that the cell area as they are scaled, at some point the core area will be equal to and then exceed the cell area. This point can be found by setting the scaled core area equal to the total scaled cell area and solving for α:

$$\alpha^2 A_{Core}=\alpha(K_1+WL) \tag{25}$$

$$\alpha = \frac{B+WL}{A_{Core}} \tag{26}$$

This is the factor by which the core **400** must be enlarged to accommodate the total cell area.

After a satisfactory placement has been found in Step **560**, the cell area in individual buckets **410** is balanced to balance routing resource usage and area usage among all buckets **410**. First, a global router assigns routes to all nets in Step **365** and an analysis of routing resources on the core **400** determines congested areas. In Step **370**, cells **420** in the most congested areas are "padded" by arbitrarily increasing their areas slightly, and cells **420** in the most underutilized areas are "shrunk" by arbitrarily reducing their areas slightly. This tends to increase the rate at which cells **420** migrate from overutilized areas to underutilized areas.

Next, a bucket equalization process is applied to the cells **420** to move cells **420** from overutilized buckets **410** to underutilized ones in Step **375**. This is a sort of "bucket brigade" movement in which a cell **420** moves at most from one bucket **410** to an adjacent bucket **410**. For example, in a series of ten consecutively numbered buckets **410** on a horizontal path, if cells **420** need to be moved from bucket **1** to bucket **10**, some cells **420** are moved from bucket **1** to bucket **2**; some from bucket **2** to bucket **3**, etc. As cells **420** move from one bucket **410** to another, the loads of nets attached to them change. This causes a corresponding change in the area of other cells **420** in the design, and these are corrected locally rather than through a global recalculation process. To ensure that changes to cell areas are minimized, cell movements along many different paths are examined and only the best used.

Finally, in the pairwise refinement process of Step **380**, a mincut process is applied between adjacent buckets **410** in

a sweeping fashion as shown in FIG. **8**. Starting from the topmost corner of core **400**, each bucket **410** and its immediate neighbors to the right and bottom are repartitioned in order to reduce the number of crossing nets. One full pass of the repartitioning ends when the bottom rightmost bucket **410** is reached. At this point, the total wire length in the circuit is computed in Step **385** and the areas of all cells **420** are readjusted in Step **390**. If there is an improvement in wire length, another iteration through the process is begun at Step **365**; if not, this part of the process is complete.

After the strengths of the cells **420** are optimized based on the timing analysis using the global routing topologies, track routing is performed which assigns a specific routing space in the bucket **410** to each wire. The final stage of physical routing software puts the physical wires following the space assignment made by the track routing as closely as possible and completes the connections for all the cells.

The present invention has been described above in connection with a preferred embodiment thereof; however, this has been done for purposes of illustration only, and the invention is not so limited. Indeed, variations of the invention will be readily apparent to those skilled in the art and also fall within the scope of the invention.

For example, the embodiment has been described in connection with a discrete integer spacing architecture; however, the invention is not so limited and may be used with fractional or other non-integer systems as well. Further, although in the preferred embodiment a maximum spacing of three was used, higher-order spacings may be used as well.

What is claimed is:

1. A method of estimating capacitance of interconnect wires in an integrated circuit chip design, the method comprising:

grouping a plurality of cells in the design into a plurality of buckets;

maintaining a congestion score for each bucket;

when routing a wire through a bucket, modifying the congestion score accordingly; and

using the congestion score to calculate an estimated capacitance for the wire.

2. The method of claim **1**, wherein the congestion score is a ratio of a number of available wire routing spaces in a given layer of the bucket in a given direction to a total number of wire routing spaces in the given layer of the bucket in the given direction.

3. The method of claim **1**, wherein maintaining the congestion score for each bucket includes excluding a wire routing space required to be empty in order to meet design rules from consideration as an available wire routing space.

4. The method of claim **1**, wherein using the congestion score to calculate an estimated capacitance for the wire includes estimating a spacing on at least one side of wires in a bucket based on that bucket's congestion score.

5. The method of claim **4**, wherein using the congestion score to calculate an estimated capacitance for the wire includes estimating a spacing on both sides of wires in the bucket based on that bucket's congestion score.

6. The method of claim **4**, wherein a range of congestion scores is equivalent to a given spacing configuration for wires in the bucket.

7. The method of claim **1**, wherein using the congestion score to calculate an estimated capacitance for the wire includes adding together a plurality of estimated capacitances for segments of the wire in different buckets.

8. The method of claim **1**, wherein the calculated estimated capacitance includes a lateral capacitance component.

* * * * *

C

US006931610B1

(12) **United States Patent**
Buch et al.

(10) **Patent No.: US 6,931,610 B1**
(45) **Date of Patent: Aug. 16, 2005**

(54) **METHOD FOR RAPID ESTIMATION OF WIRE DELAYS AND CAPACITANCES BASED ON PLACEMENT OF CELLS**

(75) Inventors: **Premal V. Buch**, Daly City, CA (US); **Manjit Borah**, Los Altos, CA (US)

(73) Assignee: **Magma Design Automation, Inc.**, Santa Clara, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/570,081**

(22) Filed: **May 12, 2000**

(51) **Int. Cl.**[7] **G06F 17/50**
(52) **U.S. Cl.** **716/5**; 716/13; 716/14
(58) **Field of Search** 716/5, 13, 14

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,461,576	A *	10/1995	Tsay et al.	716/6
5,784,289	A *	7/1998	Wang	716/8
6,151,568	A *	11/2000	Allen et al.	703/14
6,434,731	B1 *	8/2002	Brennan et al.	716/10
6,513,149	B1 *	1/2003	Donato	716/12
6,658,631	B1 *	12/2003	Pyo et al.	716/4
2001/0010090	A1 *	7/2001	Boyle et al.	716/2

OTHER PUBLICATIONS

NN9108421, "Method of Estimating Capacitance in RC Constraint Generation", IBM Technical Disclosure Bulletin, vol. 34, No. 3, Aug. 1991, pp. 421–423 (4 pages).*

Sivakumar et al., "The clade vertebrata: spines and routing in ad hoc networks", Proceedings of Third IEEE Symposium on Computers and Communications, Jun. 30, 1998, pp. 599–605.*

Das et al., "Routing in ad hoc networks using a spine", Proceedings of the Sixth International Conference on Computer Communications and Networks, Sep. 22, 1997, pp. 34–39.*

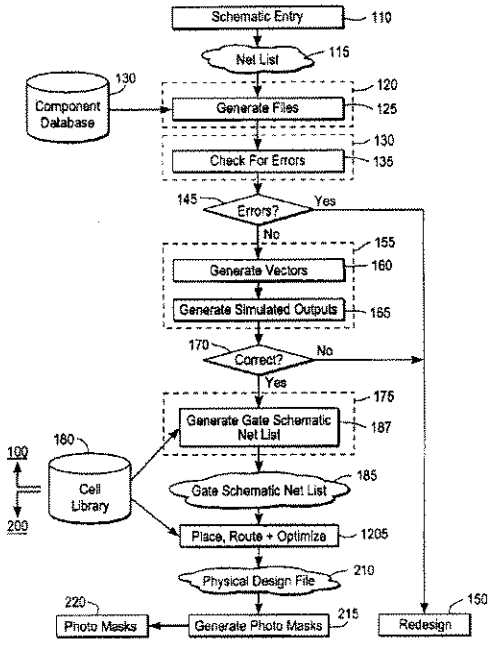
* cited by examiner

Primary Examiner—A. M. Thompson
Assistant Examiner—Phallaka Kik
(74) *Attorney, Agent, or Firm*—Pillsbury Winthrop Shaw Pittman

(57) **ABSTRACT**

A fast method of estimating capacitances and wire delays in an integrated circuit design is based on placement information such as that contained in a gate schematic net list from a logic synthesis tool. A simple tree topology called a spine tree is constructed to connect the pins of the net as an approximation of actual connections therein. Capacitance is extracted for this topology assuming a worst case scenario, and Elmore delays are computed for the wire delays based on the worst-case capacitances. The method takes linear time as a function of the number of pins in the net and is much faster than using a Steiner tree method in this context.

10 Claims, 6 Drawing Sheets



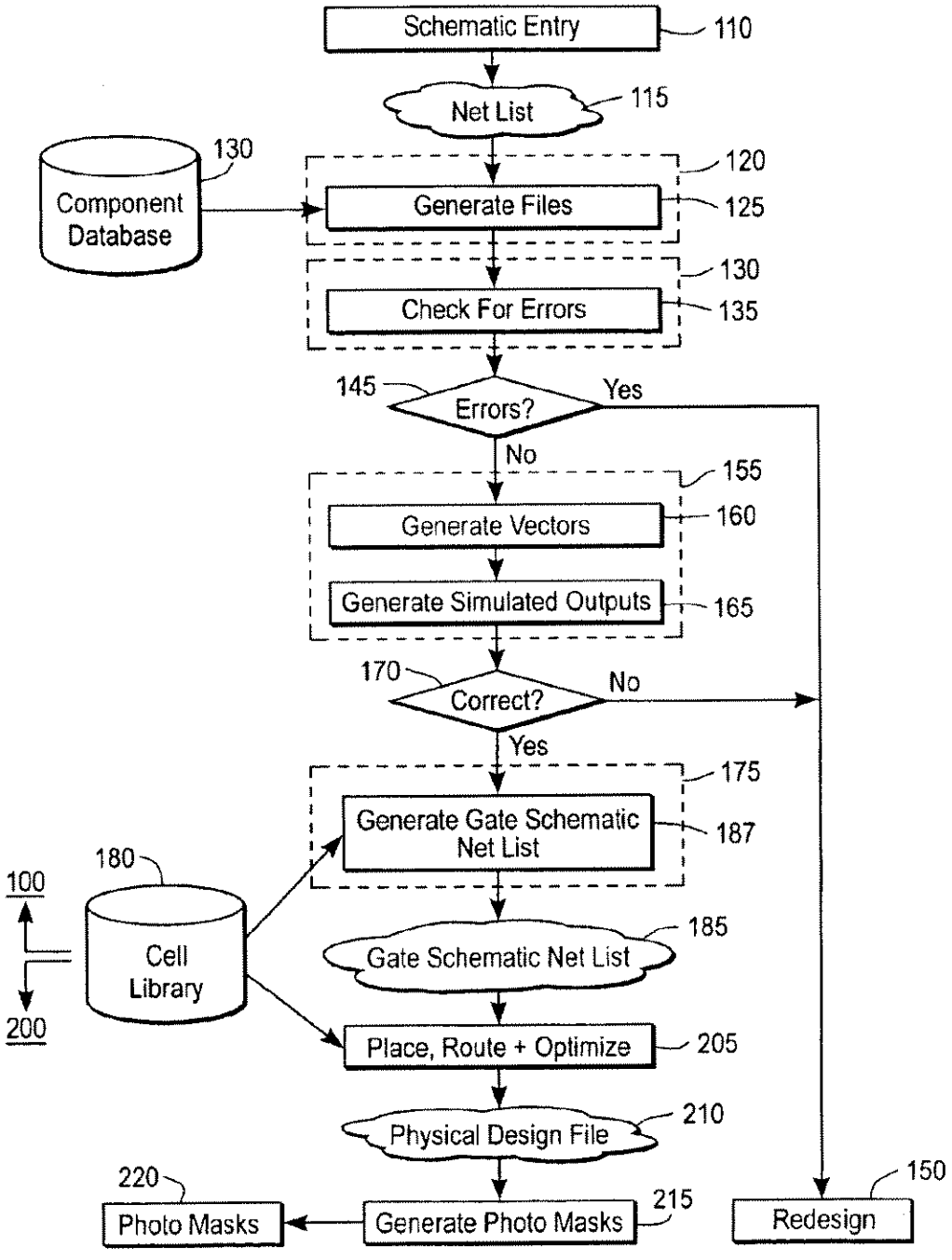


FIG. 1 PRIOR ART

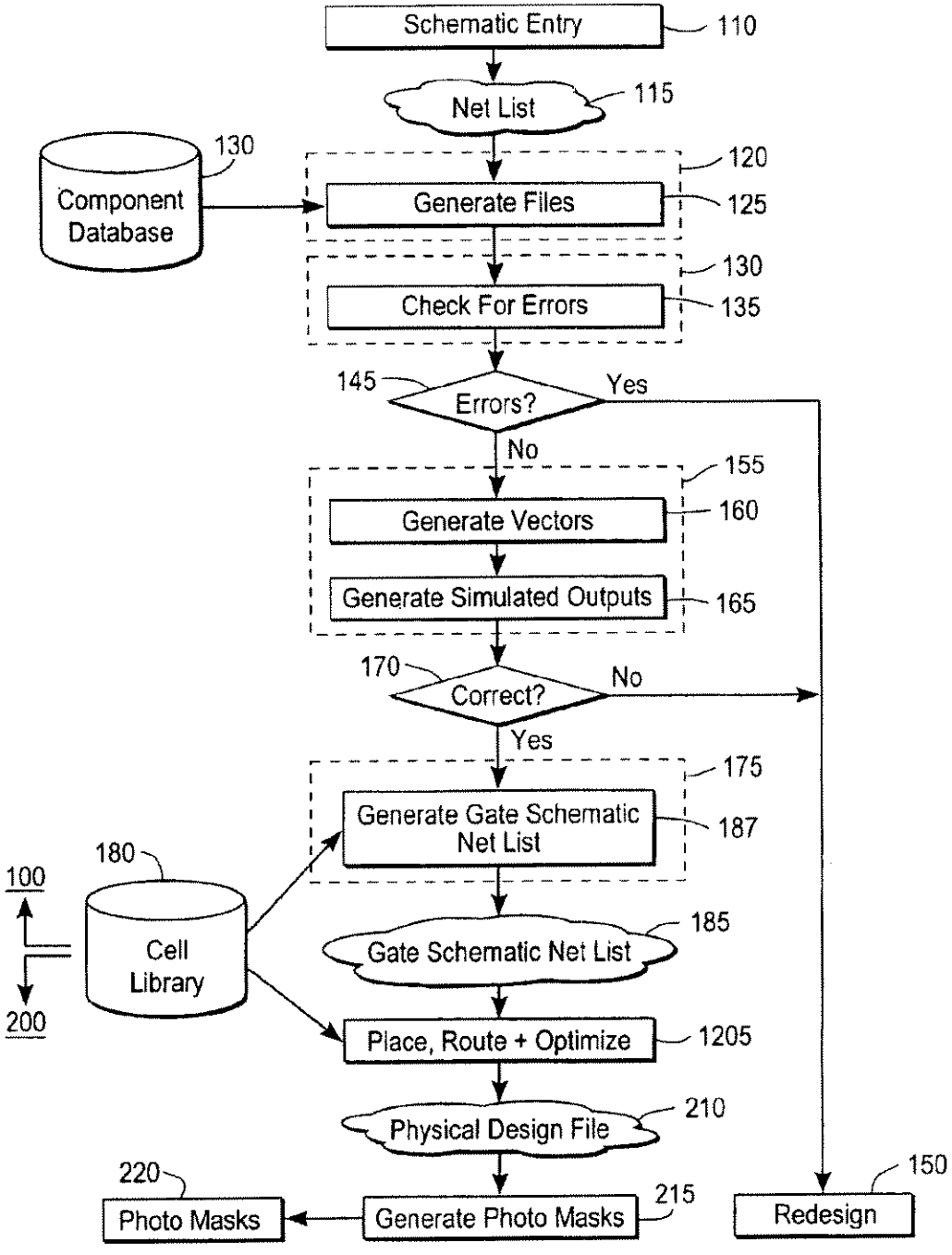

Schematic Entry
110
Net List
115
130
Component Database
120
Generate Files
125
130
Check For Errors
135
145
Errors?
Yes
No
155
Generate Vectors
160
Generate Simulated Outputs
165
170
Correct?
No
Yes
175
Generate Gate Schematic Net List
187
180
100
Cell Library
200
185
Gate Schematic Net List
Place, Route + Optimize
1205
210
Physical Design File
220
Photo Masks
Generate Photo Masks
215
150
Redesign

FIG. 2

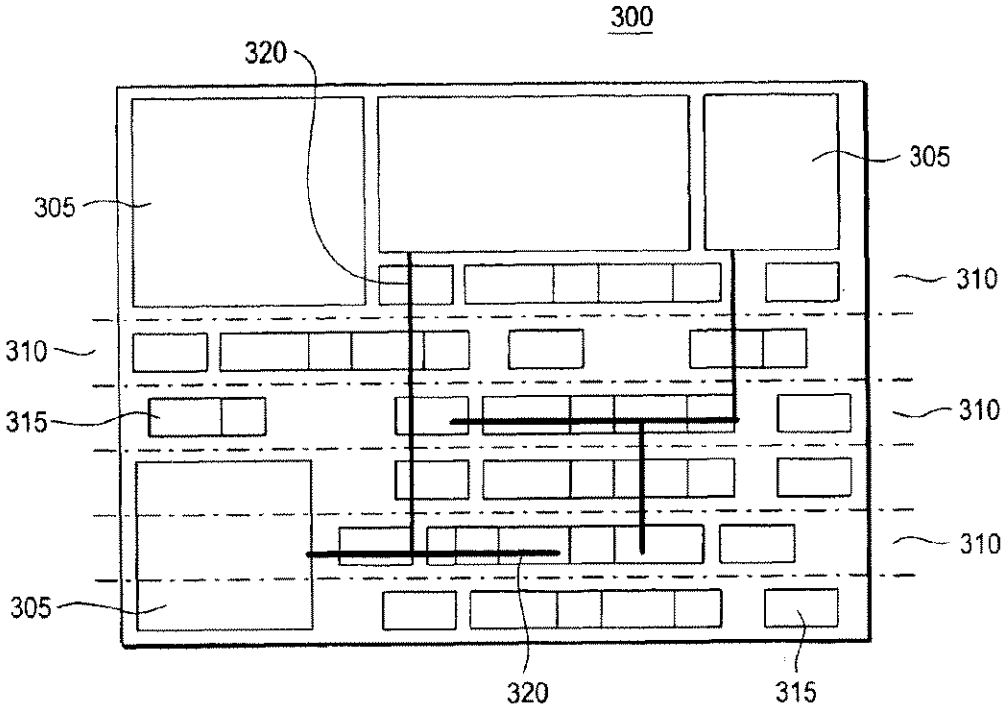


FIG. 3

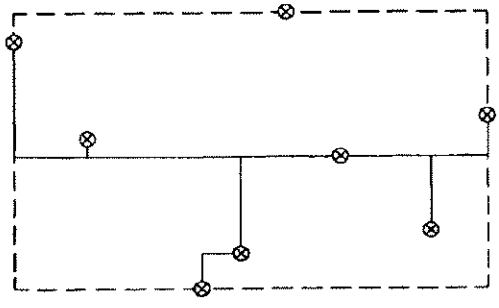


FIG. 7G

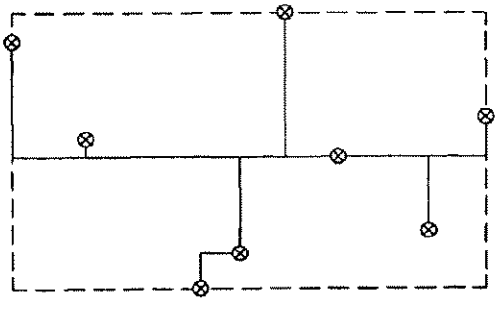


FIG. 7H

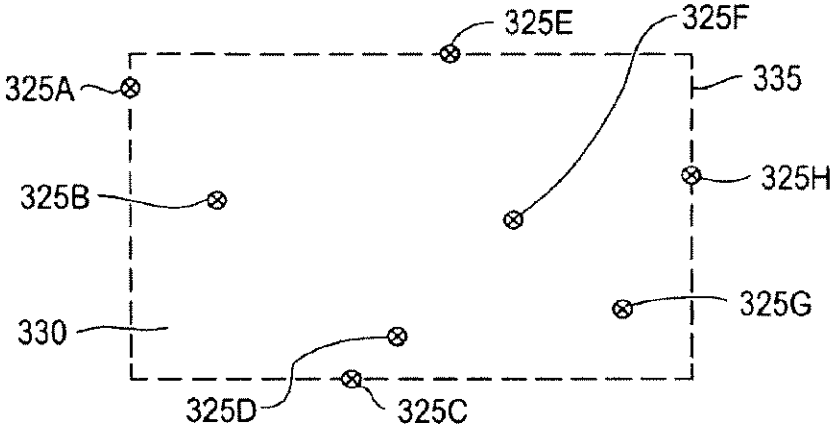


FIG. 4

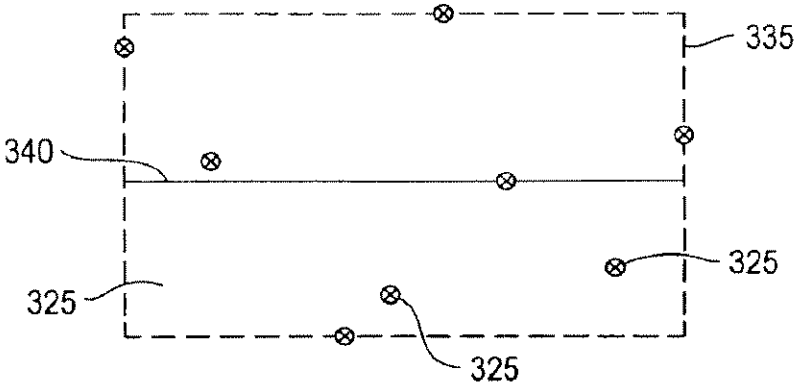


FIG. 5

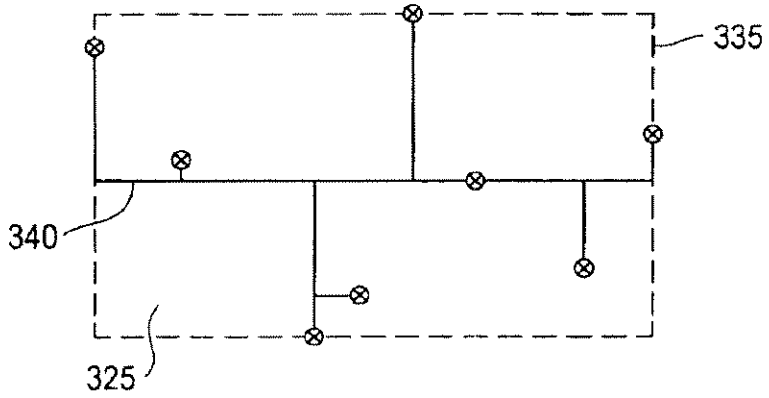


FIG. 8

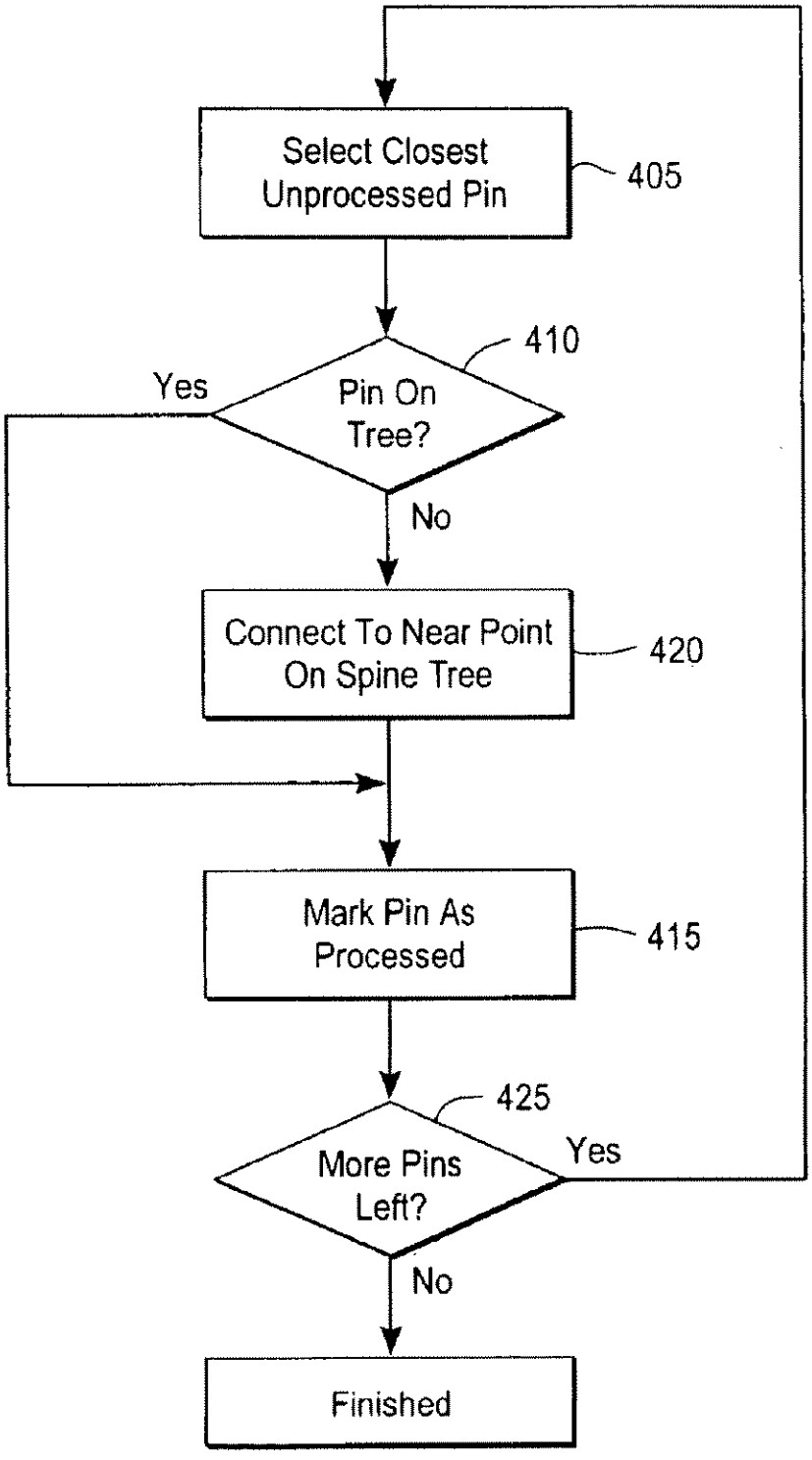

Select Closest Unprocessed Pin
405
410
Pin On Tree?
Yes
No
Connect To Near Point On Spine Tree
420
Mark Pin As Processed
415
425
More Pins Left?
Yes
No
Finished

FIG. 6

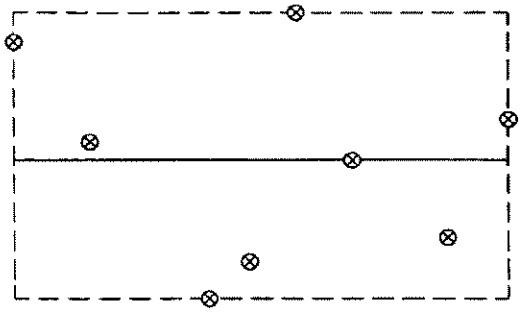


FIG. 7A

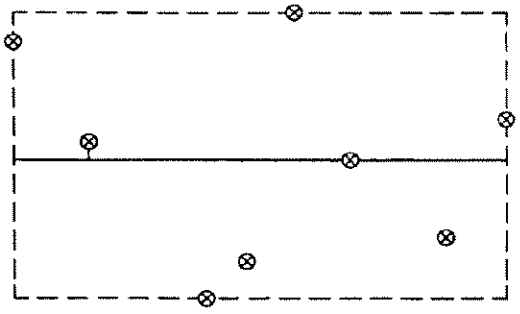


FIG. 7B

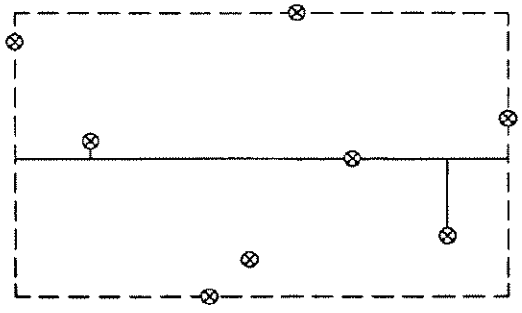


FIG. 7C

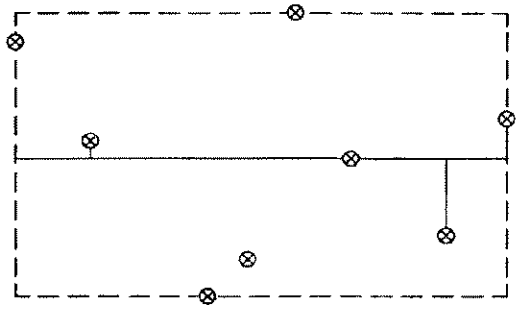


FIG. 7D

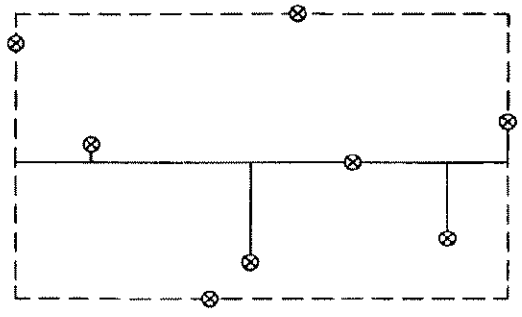


FIG. 7E

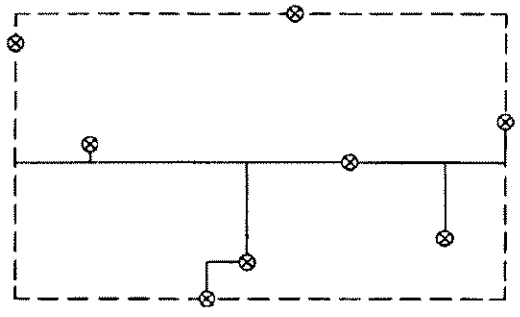


FIG. 7F

METHOD FOR RAPID ESTIMATION OF WIRE DELAYS AND CAPACITANCES BASED ON PLACEMENT OF CELLS

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention is directed to digital logic design systems. More particularly, the invention is directed to automated digital logic synthesis and placement systems for integrated circuits, and to performance optimization of digital integrated circuits.

2. Background of Related Art

Prior art computer aided design (CAD) systems for the design of integrated circuits (ICs) and the like assist in the design thereof by providing a user with a set of software tools running on a digital computer. In the prior art, the process of designing an integrated circuit on a typical CAD system is done in several discrete steps using different software tools.

The design process can be broadly divided into two phases. The initial phase **100** (shown in FIG. **1**) of selecting the right components and connecting them so that the desired functionality is achieved is called logic synthesis. The second phase **200**, in which the selected components are placed within the confines of the chip boundaries and the connecting wires are laid out in order to generate the photographic masks for manufacturing, is called physical synthesis.

First, in the logic synthesis phase **100** a schematic diagram of the integrated circuit is entered interactively in Step **110** to produce a digital representation **115** of the integrated circuit elements and their interconnections. This representation **115** may initially be in a hardware description language such as Verilog or VHDL and then translated into a register transfer level (RTL) description in terms of predesigned functional blocks, such as memories and registers. This may take the form of a data structure called a net list.

Next, a logic compiler **120** receives the net list in Step **125** and, using a component database **130**, puts all of the information necessary for layout, verification and simulation into object files whose formats are optimized specifically for those functions.

Afterwards, in Step **135** a logic verifier **140** preferably checks the schematic for design errors, such as multiple outputs connected together, overloaded signal paths, etc., and generates error indications in Step **145** if any such design problems exist. In many cases, the IC designer improperly connected or improperly placed a physical item within one or more cells. In this case, these errors are flagged to enable her to correct the layout cells in Step **150** so that they perform their proper logical operation.

Also, in Step **135** the verification process preferably checks the cells laid out by hand to determine if multiple design rules have been observed. Design rules may include the timing requirements of the circuit, the area occupied by the final design and parameters derived from other rules dictated by the underlying manufacturing technology. These design rules are provided to integrated circuit designers to ensure that a part can be manufactured with a high degree of yield. Most design rules include hundreds of parameters and, for example, include pitch between metal lines, spacing between diffusion regions in the substrate, sizes of conductive regions to ensure proper contacting without electrical short circuiting, minimum widths of conductive regions, pad sizes, and the like. If a design rules violation is identified in Step **150**, this violation is preferably flagged to the IC designer so that she can properly correct the cells so that they are in accordance with the design rules in Step **150**.

Then, using a simulator **155** the user of the CAD system may prepare a list of vectors representing real input values to be applied to a simulation model of the integrated circuit in Step **160**. This representation may be translated into a form which is better suited to simulation. This representation of the integrated circuit is then operated upon by the simulator which produces numerical outputs analogous to the response of a real circuit with the same inputs applied in Step **165**. By viewing the simulation results, the user may then determine in Step **170** if the represented circuit will perform correctly when it is constructed. If not, she may re-edit the schematic of the integrated circuit, re-compile it and re-simulate it in Step **150**. This process is performed iteratively until the user is satisfied that the design of the integrated circuit is correct.

Then, the human IC designer may present as input to a logic synthesis tool **175** a cell library **180** and a behavioral circuit model. The behavioral circuit model is typically a file in memory which looks very similar to a computer program, and the model contains instructions which logically define the operation of the integrated circuit. The logic synthesis tool **175** maps the instructions from the behavioral circuit model to one or more logic cells from the library **180** to transform the behavioral circuit model to a gate schematic net list **185** of interconnected cells in Step **187**. The gate schematic net list **185** is a database having interconnected logic cells which perform a logical function in accordance with the behavioral circuit model instructions. Once the gate schematic net list **185** is formed, it is provided to a place and route tool **205** to begin the second phase of the design process, physical synthesis.

The place and route tool **205** is preferably then used to access the gate schematic net list **185** and the library cells **180** to position the cells of the gate schematic net list **185** in a two-dimensional format within a surface area of an integrated circuit die perimeter. This process is optimized as will be explained in greater detail below. The output of the place and route step may be a two-dimensional physical design file **210** which indicates the layout interconnection and two-dimensional IC physical arrangements of all gates/cells within the gate schematic net list **185**. From this, in Step **215** the design automation software can create a set of photographic masks **220** to be used in the manufacture of the IC.

One common goal in chip design involves timing performance. The timing performance of the chip is determined by the time required for signals to propagate from one register to another. Clock signals driven at a certain frequency control storage of data in the registers. The time required for a signal to propagate from one register to another depends on the number of levels of cells through which the signal has to propagate, the delay through each of the cells and the delay through the wires connecting these cells. The logic synthesis phase **100** influences the number of levels and the propagation delay through each cell because in it the appropriate components are selected, while the physical synthesis **200** phase affects the propagation delay through the wires.

During the process of timing optimization during physical design in Step **205**, circuit timing is evaluated based on an initial placement and selection of cell strengths. The feedback from the timing analysis is used to drive repeated improvements to the placement software and the selection of the strengths of the cells. The automation software may also

perform buffering on some parts of the circuit to optimize the timing performance by inserting repeater cells, i.e., buffers, to speed up certain paths. See, for example, U.S. patent application Ser. No. 09/300,557 to Buch et al., now U.S. Pat. No. 6,553,338. Preferably, the optimization software tentatively applies one such modification, evaluates the timing and other constraints (such as design rules dictating capacitance limits) to determine if the step is acceptable and then makes the change permanent if it is deemed acceptable.

The number of changes to the circuit in terms of change in placement and net list can be numerous during the automatic optimization step **205**. In order to evaluate the timing performance and capacitance the automation software needs to construct some reasonable topologies for connecting the nets so that it can make estimates of net capacitances and, from these, timing estimates. A complete routing of all the wires at this stage is very time consuming and impractical. As an alternative, Steiner tree topologies may be used successfully to estimate delays based on placement. However, constructing a Steiner tree also requires an algorithm with quadratic complexity which makes the optimization process very slow and impractical for a large circuit. While it is important to have accurate estimates of the capacitance and delays during the placement and net list optimization step **205**, it is also important to be able to compute the delays and capacitance as quickly as possible. Computing the capacitance of a net takes at least linear time as a function of the number of pins in the net. Therefore a linear time algorithm to compute the capacitance is desirable.

SUMMARY OF THE INVENTION

The present invention has been made in view of the above-described problems of the art, and it is an object of the present invention to provide a fast method of estimating capacitances and wire delays in an integrated circuit design based on placement information such as that contained in a gate schematic net list from a logic synthesis tool. A simple tree topology called a spine tree is constructed to connect the pins of the net as an approximation of actual connections therein. Capacitance is extracted for this topology assuming a worst case scenario, and Elmore delays are computed for the wire delays based on the worst-case capacitances. The method takes linear time as a function of the number of pins in the net and is much faster than using a Steiner tree method in this context.

BRIEF DESCRIPTION OF THE DRAWINGS

These and other objects, features and advantages of the present invention are better understood by reading the following detailed description of the preferred embodiment, taken in conjunction with the accompanying drawings, in which:

FIG. **1** is a flowchart of an integrated circuit design process according to the prior art;

FIG. **2** is a flowchart of an integrated circuit design process according to a preferred embodiment of the present invention;

FIG. **3** shows an example circuit having large custom blocks and rows of standard cells for use with the preferred embodiment;

FIGS. **4** and **5** show elements of a spinal network in the preferred embodiment;

FIG. **6** shows an algorithm used in constructing the spinal network in the preferred embodiment;

FIGS. **7A–7H** show a step-by-step process of developing a spinal network in the preferred embodiment; and

FIG. **8** shows a complete spinal network in the preferred embodiment.

DETAILED DESCRIPTION OF PRESENTLY PREFERRED EXEMPLARY EMBODIMENTS

Returning to the automated IC design process described above, a preferred embodiment of the present invention is preferably incorporated as part of a place and route tool **1205** as shown in FIG. **2**. FIG. **3** shows a circuit **300** with large blocks **305** and rows **310** of standard logic cells **315** which might be produced by the place and route tool **1205** as is known in the art. In the preferred embodiment, the place and route tool **1205** preferably optimizes timing of the circuit **300** by choosing the placement of the blocks **305** such that the delay across the interconnect wires **320** between cells **315** is minimized and the cells strengths are sufficient to guarantee the required timing performance. The optimization adds repeaters or buffers to selected wires to speed up certain paths. Preferably, placement of the cells, selection of the strengths and insertion of buffers are done in a single optimization process to evaluate the best possible solution.

The place and route tool **1205** then preferably operates as a router to determine the placement of interconnect wires **320** connecting the appropriate signals within the circuit boundary and between the various cells **315**. Using the net list thus obtained (which maybe different from the original net list due to insertion of buffers or repeaters) and the placement and routing generated as described above, as noted above the design automation software develops a mask layout **220** to finally fabricate the circuit. The net list thus obtained can also be used by other automation software to verify the functionality of the circuit by comparing it with the original net list that has been certified to perform the required functionality.

The present invention may be used to evaluate the wire delays and capacitances quickly during the process of optimizing the placement, selection of cell strengths and insertion of buffers. Every change in the placement changes the position of the pins and hence needs the capacitance and delays to be re-evaluated. When a buffer is inserted, a new net is created and the original net is changed, thus causing two nets to be evaluated. Similarly, when the size of a cell is changed to a different strength, the new cell may have a different shape and area which will cause the pin positions to change, requiring recalculation of the nets connected to all the pins of the cell.

The present invention can be used to recompute the capacitance and wire delays quickly during this optimization process.

As described above, for optimization purposes it is useful to quickly estimate wire delay times. For this purpose, it is useful to quickly estimate interconnect wire capacitances. In order to compute the capacitance and delay of interconnect wires **320**, the positions of the pins **325** of a net **330** in the placed design are used to draw a bounding rectangle **335** for the net **330** as shown in FIG. **4**. As is known in the art, pins are the points in the layout of a predesigned standard block or a custom block where a wire can be physically connected to hook it up into the signal net. Preferably, the pins are positioned on the boundaries of the block.

This bounding rectangle **335** is the smallest rectangle which encloses all the pins **325** of the net **330** and that lies within the boundary of the chip. The main purpose of the bounding rectangle **335** is to determine whether the pins **325**

are generally laid out in a horizontal or vertical direction, thereby enabling an efficient arrangement for the spine.

Preferably and with reference to FIG. **4**, borders of the bounding rectangle are determined by identifying which of the pins **325** is the leftmost (using the drawing page as a frame of reference); which of the pins **325** is the rightmost; which of the pins **325** is uppermost; and which of the pins **325** is lowermost. Sides of the bounding rectangle are then defined by horizontal lines passing through the uppermost and lowermost pins (in the case of FIG. **4**, pins **325**E and **325**C, respectively) and vertical lines passing through the rightmost and left most pins (here, pins **325**B and **325**H).

The shape of the bounding rectangle **335** is used to determine the direction of the "spine" of the net **330**. For example, the rectangle in FIG. **4** is longer in the horizontal direction; hence, the spine **340** for this net **330** will be horizontal.

Once the direction of the spine **340** is determined, its location within the bounding rectangle **335** is determined by taking the average of one of the Cartesian coordinates of all pins **325**. If the spine **340** has been determined to be horizontal, the average of the y-coordinates of the pins **325** is taken; if the spine **340** has been determined to be vertical, the average of the x-coordinates of the pins **325** is taken.

Consider the bounding rectangle **335** and pins **325** within net **330** of FIG. **4**. Suppose the bounding rectangle **335** is mapped onto a 25×15 Cartesian grid so that the coordinates of the pins **325** are as follows, moving from left to right:

325A: (0, 13.75)	325E: (14.375, 15)
325B: (3.75, 6.25)	325F: (16.875, 7.5)
325C: (10, 0)	325G: (21.875, 3.125)
325D: (11.875, 1.875)	325H: (25, 9.375)

Then, averaging the y-coordinates we find that the spine **340** is at y=7.1—slightly below the middle of the bounding rectangle **335**.

FIG. **5** shows this horizontal spine **340**. Once the spine **340** is constructed, all the pins **325** are connected to it using the algorithm **400** shown in FIG. **6**. First, the pin **325** closest to the spine tree **345** (see FIG. **8**), i.e., the spine **340** and all connections from it to pins **325**, is identified in Step **405**. If this pin **325** μdies on the spine tree **345** in Step **410**, it is marked as processed in Step **415** and execution continues as described below. If not, Step **420** connects the pin **325** to the nearest point on the spine tree **345** and the pin **325** is marked as processed in Step **415**. If Step **425** determines that there are any unprocessed pins **325** left, execution returns to Step **405**; otherwise, the spine tree **345** for this net **330** has been developed and may be used for capacitance and delay estimates as described above.

FIGS. **7**A–**7**H show the step-by-step process of building the spine tree **345** for net **330** according to the algorithm shown in FIG. **6**.

FIG. **8** shows the final spine tree **345** constructed for the set of pins **325**. The capacitance for the spine tree **345** may be computed by adding the capacitance of each segment thereof together. The capacitance for each segment is calculated using the parasitic extraction rules which are measured from the fabrication process. These rules are expressed in terms of the capacitance due to the overlap in the vertical direction with other layers of metal (i.e., area capacitance) and the overlap with other wires in the same layer (i.e., lateral capacitance). In order to be conservative in the estimates a worst-case scenario is adopted where it is assumed that there is a wire on both sides of the wire with minimum allowable distance between them.

The delays are computed using the Elmore delay formula based on the spine topology. This technique is known in the art and explained in great detail in Elmore, "The Transient Response of a Damped Linear Network with Particular Regard to Wideband Amplifiers", J. Applied Physics, 19:55–63 (1948) (incorporated herein by reference), and for simplicity and brevity will not be described further here. These delays may then be used for calculating estimates in optimizing the placement and routing of the cells as described above.

The present invention has been described above in connection with a preferred embodiment thereof, however, this has been done for purposes of illustration only, and the invention is not so limited. Indeed, variations of the invention will be readily apparent to those skilled in the art and also fall within the scope of the invention.

What is claimed is:

1. A method of estimating capacitance of interconnection wires in an integrated circuit design comprising:

determining a rectangle bounding a group of pins whose capacitance is to be estimated;

determining a direction of a spine for connecting the pins based on the bounding rectangle;

connecting the pins to the spine to minimize a total length of the spine and connections, thereby forming a spine tree; and

using the spine tree as a parameter for estimating a capacitance of the interconnection wires.

2. The method according to claim **1** wherein the step of determining a direction of the spine also determines a location of the spine with the bounding rectangle.

3. The method according to claim **1** wherein the step of using the spine tree as a parameter for estimating the capacitance of the interconnection wires adds together capacitances of each different segment that make up the spine tree.

4. The method according to claim **3** wherein the capacitance for each segment is calculated using parasitic extraction rules.

5. The method according to claim **4** wherein the parasitic extraction rules adopt a worst case scenario of wire overlap.

6. The method according to claim **1** further including the step of calculating wire delays.

7. The method according to claim **6** wherein the step of calculating wire delays uses an Elmore delay formula based on a topology of the spine tree.

8. The method according to claim **7** wherein the step of using the spine tree as a parameter for estimating the capacitance of the interconnection wires adds together capacitances of each different segment that make up the spine tree.

9. The method according to claim **8** wherein the capacitance for each segment is calculated using parasitic extraction rules.

10. The method according to claim **9** wherein the parasitic extraction rules adopt a worst case scenario of wire overlap.

* * * * *